ifas Institut für fluidtechnische Antriebe und Systeme | RWTH AACHEN UNIVERSITY

This work was submitted to the

Institute for Fluid Power Drives and Systems (IFAS)
RWTH Aachen University
Univ.-Prof. Dr.-Ing. Katharina Schmitz

Master Thesis

Dimension Reduction of Variable Time Period Datasets of a Pneumatic System with Deep Unsupervised Learning

by

Tzu-Chen Liu, B.Sc.

Matr.-No. 404756

December 2021

Supervisors:
Univ.-Prof. Dr.-Ing. Katharina Schmitz
Faras Brumand-Poor, M.Sc. M.Sc.

Research Field:
Digitalization

Eidesstattliche Versicherung

,	
Name, Vorname	Matrikelnummer

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit mit dem Titel

Dimension Reduction of Variable Time Period Datasets of a Pneumatic System with Deep Unsupervised Learning

selbständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Aachen, den	
Ort, Datum	Unterschrift

Belehrung:
§156 StGB Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.
§161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt
(1) Wenn eine der in den §§154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des §158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

Aachen, den	
Ort, Datum	Unterschrift

Aufgabenstellung

Fluidtechnische Systeme sind meistens mit einer Vielzahl von Sensoren und Aktoren ausgestattet. Diese erlauben neben der Überwachung des Betriebs auch ein Eingreifen, um ein gewünschtes Prozessverhalten zu erreichen. Die Zustandsüberwachung ermöglicht die Vorhersage von Ausfall oder Fehlerfunktion der einzelnen Komponente oder sogar der gesamten Anlage. Für die Charakterisierung von fluidtechnischen Systemen wird eine große Anzahl von Betriebsdaten genutzt. Insbesondere die variable Länge von unterschiedlichen Betriebszuständen erschwert eine verallgemeinerte Betrachtung der Maschinendaten. Im Rahmen dieser Arbeit wird ein datenbasierter Ansatz aus dem Deep Learning genutzt, um genau dieses Problem zu untersuchen. Dabei werden die Betriebsdaten eines pneumatischen Prüfstands genutzt. Es wird eine Methodik aus dem Unsupervised Learning, welche den hochdimensionalen Zustandsraum der pneumatischen Anlage, untersucht. Insbesondere gilt es die variable Sequenzlänge der Betriebsdaten zu verallgemeinern, so dass der Algorithmus unabhängig von der Eingangssequenz ist. Dieser Algorithmus wird sowohl getestet als auch validiert. Anschließend werden die Parameter durch trial and error optimiert. Das letzte Arbeitspaket der Arbeit ist die schriftliche Dokumentation und die Präsentation im Plenum.

Contents

1 Introduction

In this master's thesis, a simulation dataset of a pneumatic machine is investigated. The focus is on the approach of extracting information out of the data, and the eventual goal is to identify the leakage status of this pneumatic machine based only on the reduced data.

The data consist of time-series sequences with varying lengths, which is a challenge addressed by the "windows" method proposed by this thesis. Unsupervised learning models, autoencoder (AE) and variational autoencoder (VAE) models, are trained to compress the data sequences. Different structure and parameter designs of these models are carried out and the results are presented. The data representations generated by the UL models are further investigated and are used to train leakage classifier models. Random forest and multi-layer perceptron classification algorithms are used, and high accuracy is achieved.

1.1 Motivation

Fluid power systems and applications are everywhere in our life, and it would be helpful to know in advance when these machines may malfunction, or to detect failures before more users are affected. This is the goal of condition monitoring. Condition monitoring is to detect potential faults by monitoring one or several parameters in a machine. Most fluid power systems have been equipped with a vast amount of sensors, which results in a high volume of collected data. The amount of data is too huge that the important information behind the data cannot be directly observed. Therefore, it is desired to have approaches to reduce the data reasonably.

Dimension reduction tasks, as well as all kinds of data analysis and modeling cases, data preprocessing and model tuning are necessary. However, these tasks often required customization to a specific application. In another word, a trained model or a training process is subject to the machine that the dataset comes from. If it is desired to perform condition monitoring on another machine, the new dataset needs to be preprocessed based on its own characteristics. Furthermore, for most researches in the field of condition monitoring, the input data sequences have the same or similar length. This restricts the established models and pipelines from those data with varying sequence lengths.

The motivation of this thesis is to yield a more general and less customized training pipeline that can be used to compress the input data. As fewer data preprocessing procedures are carried out as possible, and the model can also take data with varying sequence lengths as input. Moreover, the size of the outcome of the models is drastically reduced, but still meaningful enough to identify the leakage status of the target machine.

1.2 Related Work

Data-based condition monitoring applied to fluid power systems has been researched in recent years. The applications can be early detection of failures and predictive maintenance [1]. In order to secure the safety and reliability of technical processes, faults should be able to be detected, diagnosed, and managed [2]. Different aspects of condition monitoring are under investigation. Some researchers focus on real data collected from test rigs, others used a simulation method to collect enough amount of data for their research because experimental data can be expensive to obtain, e.g. Makansi et al. investigated on simulation-based data sampling [3]. Some researchers used classical data analysis methods such as signal preprocessing, Fourier transform, statistical analysis, for example, Kovacs et al. used signal preprocessing approaches to monitor the behavior of a pneumatic actuator [4]. Others used machine learning or even deep learning techniques for classifying failure situations. The neural network is a widely adopted supervised machine learning technique in this field. Karpenko et al. classified faulty operation of pneumatic control valves via neural network pattern classifier [5]. Demetgul et al. compared two different neural network designs for classifying faults of a pneumatic system [6]. Sano et al. compared deep learning and support vector machine, a classical machine learning algorithm, on health monitoring and prognosis of aircraft systems [7]. Lurette et al. proposed neural network and decision tree models that can perform on-line monitoring [8].

Instead of emphasizing the fault classifiers, some researchers built the model of the target physical system and identify anomalies by comparing real data with their models. Münchhof used model-based algorithms to detect faults in linear hydraulic servo-axis [9]. Zachrison established self-organising maps, which is an advanced neural network, to model the physical system and to locate the weak part of a component by observing the model [10]. Wang et al. modeled the system with a radial basis neural network and assessed the performance by the Mahalanobis distance between the observed data and the prediction made by the model [11]. Siyuan et al. built a model of the target system by Principal Component Analysis of Q statistics [12].

One challenge of data-based condition monitoring is the size of data. More data provide more

information, but enormous data also cause problems such as long training time, high computational resources required, including a high amount of outliers, and difficulty for models and for humans to comprehend. There are statistical approaches to reduce the dimension of input data. Duensing et al. took pressure signals as input, making failure detection a supervised learning classification problem, and the input data was reduced by feature selection and extraction [13]. Chawathe used skewness and the discrete Fourier transform to select features before classification [14]. Jegadeeshwaran et al. investigated vibration signals of hydraulic brakes [15]. Descriptive statistical features were extracted, and feature selection was done by the decision tree algorithm. Helwig et al. detect faults of the hydraulic system as well as sensors by a statistical method as well [16], and Schneider et al. compare statistical feature extraction approaches on cyclically recorded time-series data [17].

In addition to traditional feature extraction and selection techniques, unsupervised machine learning algorithms can also be used to decrease the size of data. Instead of classic feature selection approaches, Lei et al. implemented an unsupervised learning neural network to learn features of vibration signals directly [18]. Krogerus et al. use self-organized map approach [19] [20]. Huijie et al. implemented autoencoders on reducing vibration signals [21]. Mallak et al. proposed LSTM autoencoders approach for dimension reduction before performing the classification task [22]. For most researches that took time-series data as input, the length of each data sequence is the same. In the research of Hennig et al., the variational autoencoder was used, and the sequence lengths are not the same [23].

1.3 Goal

The main goal of this thesis is to monitor fault conditions on a fluid power system with a more general approach. To achieve this purpose, several sub-goals need to be fulfilled:

1. Accept time-series data sequences with different lengths
2. Reduce dimensions of machine data
3. Show that the data representation is meaningful
4. Establish a general model training pipeline that can be used on other applications

For most other researches done on condition monitoring with time-series data, the sequence length is restricted to be the same or at least similar. By achieving the first sub-goal, the models proposed by this thesis are open to more use cases. The second sub-goal is necessary for the analysis of big data. The third sub-goal is used to prove that the compressed data is although small but still contains

important information. With the fulfillment of the last sub-goal, the time and efforts of performing condition monitoring on other applications can be reduced.

1.4 Contribution

This research presents a novel method to compress enormous time-series and variable-length data. The goal is to reduce the data drastically, but the remaining information is enough to represent the original data and can be used to identify the leakage status of this pneumatic system. The data used in this research come from a pneumatic system that monitors the position and valve status. Because of the different lengths of data sequences, the input data are first processed via the ”windows” method proposed by this research. Unsupervised learning (UL) models autoencoder (AE) and variational autoencoder (VAE) are used to extract information out of the processed data. Because of the high compression rate (84% to 99.5%) on the original data, the extracted data may not be intuitive and clearly explainable. Further investigations are carried out in order to have a deeper insight into the meaning of the reduced data. Supervised learning (SL) classification models are then trained on this data representation of the original data, and can eventually classify the leakage status of the target fluid machine with high accuracy.
The contributions in this thesis can be listed as:

1. ”Windows” approach for input with varied size
2. UL models that perform high compression rate
3. Investigation on the black-box training process and latent space
4. A general model training pipeline

The first point is significant for using a dataset with variable lengths for researches. Most machine learning and deep learning models take only data sequences with the same length as input. With this ”windows” approach, the input data can be more general. A well-shaped dataset is no longer strictly required. Experimental data are often expensive to obtain. Therefore, it would be more efficient to utilize as much accessible data as possible.

The second point shows that the dimension reduction models proposed by this thesis can compress data to be only 0.5% to 16% of its original size. Furthermore, these unsupervised learning models do not require hyperparameter tuning, and can be adopted by other researches. This is useful for future researches in the field of data-based condition monitoring.

The third point is an approach of analyzing the black-box part of machine learning. The reduced dataset has different dimension, length, and other characteristics, and cannot be clearly depicted and be intuitively comprehended. How do these compressed data look like? What is the relationship between them and the original data? Why can they lead to high accuracy when classifying leakage status? These doubts can be resolved by investigation on the latent space of models.

The forth point enables the adoption of the training pipeline in other researches. In this thesis, both the ”windows” algorithm and the UL models don’t required customized tuning. As a result, the training procedures can also be applied to data from other machines, and the input data do not have to be in the same shape.

To recapitulate, this thesis has implemented UL models to reduce the time-series data with a high percentage. The input data do not have to be in any specific shape, and the outputted compressed data can be used to identify leakage status of the target machine. The data representation are investigated as well to provide deeper insight into this research.

2 Fundamentals of Machine Learning

In this chapter the fundamentals of machine learning (ML) are introduced, including an overview of ML (Sec. 2.1) and its categories supervised learning (SL) (Sec. 2.2), unsupervised learning (UL) (Sec. 2.3), and reinforcement learning (RL) (Sec. 2.4). In SL tasks, targets or labels are provided, and common applications are classification and regression. In UL problems, on the other hand, there are no targets or labels in the dataset, and the goal is to dig out the similarities between data points or the important information. For RL cases, agents learn to perform a task by the rewards received from the interaction with the environment.

2.1 Overview of Machine Learning

ML consists of algorithms that use data in order to improve their desired behavior [24]. Applications for ML are classification, regression, clustering, and dimension reduction. Given a challenge and data, models are trained with suitable algorithms, in order to perform the task as well as possible.
ML has a wide range of applications, such as speech recognition, automatic translation, face detection, image recovery, and customized advertisement. In recent years, the field of ML is intensively investigated and developed the potential to be applied to countless different products and services. However, ML is actually not a new concept. Most algorithms discussed in this chapter were already proposed decades ago. Thanks to the development of computers, storage devices, and the internet, implementing ML models nowadays is not restricted to huge computer rooms anymore. Instead, everyone with a personal computer can perform ML tasks. Computing clusters for advanced ML problems are also offered by most academic institutes and also some industrial companies.
A better-organized dataset results in a better model. In the data preprocessing phase before building models, outliers are filtered out and the dataset is normalized. Outliers can be detected via domain knowledge (e.g. sensor characteristics) or anomaly detection. Data normalization converts each feature of the dataset to the same scale but retains the relative distances between data points. If features are not on the same scale, features with greater magnitudes will be more dominant and have stronger effects on the model.
The algorithm should be carefully chosen based on the characteristics of the data and the task. For classification or regression problems, SL algorithms are used. For clustering or dimension reduction

goals, UL algorithms are used. For problems where the algorithm is required to interact with a given environment, reinforcement learning algorithms are selected. Afterward, models can be trained by the selected algorithm with the dataset.

Loss functions are the metric to evaluate the performance of the model, which should also be carefully chosen to match the model's characteristics and the problem to be solved. After the loss is calculated, the model learns by adjusting its variables using the gradients of the loss with regard to every variable. The gradients are calculated by backpropagation method, which computes the gradients by the chain rule.

Overfitting is one challenge of ML. Overfitting occurs when the algorithm learns the data by heart, meaning the model has good performances when applied to this dataset, but poor performances when applied to data not included in the particular dataset. One of the common solutions to this problem is to split the dataset into training and testing parts. As a common practice, the training dataset is composed of around 70 to 80 $percent$ of the original dataset, while the testing dataset is around 20 to 30 $percent$. The model is only trained with the training dataset because the testing dataset is preserved for evaluating the model. In both the training and evaluating process, the losses are calculated according to chosen loss functions. A sign for overfitting is when the loss of the evaluation process is much higher than the loss of the training process and modifications should be conducted. The traditional workflow of a typical ML task is displayed in fig. 2.1

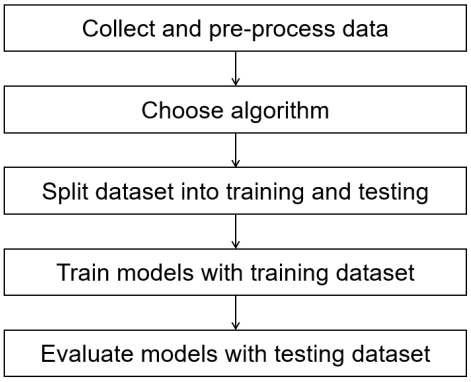


Figure 2.1: An illustration of a general workflow for deploying machine learning tasks. Starting with collecting and preprocessing data, followed by choosing a suitable algorithm, splitting the dataset to training and testing, training the model only with the training dataset, and eventually evaluating the model with a testing dataset.

2.2 Supervised Learning

Marsland defined supervised learning as "A training set of examples with the correct responses (targets) is provided and, based on this training set, the algorithm generalizes to respond correctly to all

possible inputs" [25]. In other words, the targets or the correct answers are provided by the dataset. A good model means, when receiving input, the model can generate "predictions" that match the target. The model can be evaluated by comparing the targets to the predictions. When the predictions are categories, it is a classification problem, while for continuous real values, it is a regression problem [26].

2.2.1 Classification

The goal of a classification problem is to be able to predict which category does the input data belong to, e.g. classification of a human, a rabbit, or a car on an image. For binary classification cases, the goal would be to predict whether the input data belong to a certain category or not, for example, is this vibration signal profile generated by a well-functioning or a malfunctioning car. Commonly used algorithms for classification problems are Bayes decision theory, k-nearest neighbor, support vector machine, and decision tree.

The main focus in this research is not classification, but two classification models are used as a demonstration of application: random forest (RF) and multi-layer perceptrons (MLP). RF consists of decision trees, and the classification result is the one that most of its decision trees agree on [27]. MLP, as its name suggests, contains layers of perceptrons. One perceptron is one binary classifier, and a MLP organizes its perceptrons to form a multi-classes classifier [28].

2.2.2 Regression

Different from classification models, regression models make predictions of continuous values instead of only categories. An example of a regression task is illustrated in fig. 2.2. The training dataset is generated by an unknown function, as depicted by the blue dots. The algorithm should learn the pattern of the unknown function from the input and output values. After training, the model, represented by the green line, can make predictions that mimic the unknown function. The input values for making predictions are not necessary to be among the training dataset. In this example, both input and output are only one dimension, but more complicated regression problems with more dimensions also exist.

2.3 Unsupervised Learning

Marsland defines unsupervised learning as "Correct responses are not provided, but instead the algorithm tries to identify similarities between the inputs so that inputs that have something in common are categorised together" [25]. In other words, the dataset does not include any labels or targets, and the UL algorithms don't predict the targets, but instead group data points with similarities together,

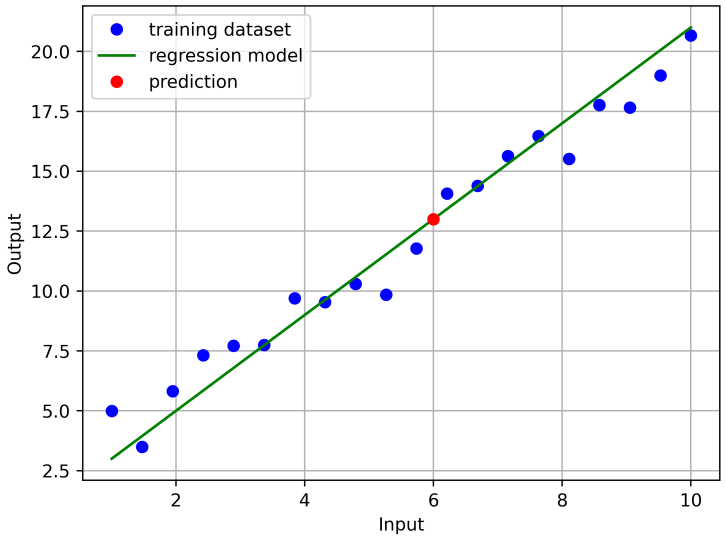


Figure 2.2: An example of linear regression. The x-axis is the input values, and the y-axis is the corresponding output values. Blue dots represent data points in the training dataset, and the green line stands for the regression model trained by the training dataset. The input values do not include 6, but the model can still give a prediction of the case when input equals 6, as the red dot shows.

also called clustering, or extract the crucial information out of the dataset as in dimension reduction tasks.

2.3.1 Clustering

Clustering algorithms can be used when the goal is to group similar items in the same clusters as well as less similar items in separate clusters. An example of a clustering result is displayed in fig. 2.3. The three families of clustering algorithms are partitioning clustering, hierarchical clustering, and density-based clustering [26]. Partitioning clustering algorithms learn the cluster centers and predict every input data point to a cluster whose center is the nearest. Hierarchical clustering algorithms initiate small clusters with a more than required amount and find clusters by finding hierarchy and merging small clusters. Density-based clustering algorithms locate high-density regions and separate clusters with low-density regions. The labels of data in many real-world problems are not available, and clustering algorithms are therefore very helpful. Anomaly detection and data segmentation are two common application scenarios for clustering algorithms.

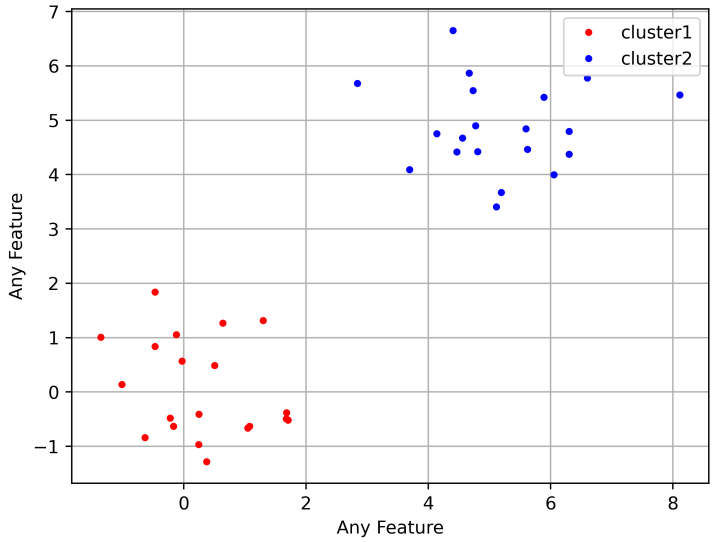


Figure 2.3: An example of clustering. In this example, the same dataset is used for training and making predictions. The x- and y-axis can be any feature in the dataset, and every dot represents one data point. The colors stand for labels, which are not provided by the dataset but the prediction. After training, the clustering model outputs labels of each data point, showing which cluster should each point belong to.

2.3.2 Dimension Reduction

When the dataset consists of more features, it contains more information that might better help the model to learn. However, the computation time grows exponentially with the increase of dimensions [29]. Therefore, if a feature can only add little information to the dataset, the tiny increment of information may not be worth the exponentially increased computation time. The trade-off between the amount of information and the computational resources is always a challenge, and the goal of dimension reduction is to find an optimal point in between by extracting the more crucial information among all data.

Principal Component Analysis

When features in a dataset are highly correlated, some features can only bring little information to the dataset, and it may be possible to use fewer features to represent a similar amount of the information. In extreme cases, some features may even be completely redundant, e.g. one feature is another feature plus a constant. If a feature has a very low variance, meaning its value doesn't change much when the other features and labels change, maybe this feature is not very relevant to this application scenario. The extreme case is that all values in this feature remain the same value. Based on these concepts,

principal component analysis reduces feature dimensions by generating new uncorrelated features, which also maximize the variances [30]. In this way, dimensions are reduced, but more important and more relevant information is retained.

Autoencoder

The idea of the autoencoder is to encode data into smaller sizes [31]. From a different point of view, to extract the most important information out of the data. However, unsupervised learning means there are no given labels. How can the autoencoder model be evaluated without labels? How can the models know whether it encodes the data wisely or not?

Autoencoder consists of an encoder and a decoder as shown in fig. 2.4. Both the encoder and decoder consist of one or multiple layers of neural networks. The encoder encodes data into a latent space, and then the decoder takes the values in latent space as input and aims to reconstruct the input data. The goal is to make the reconstructed data the same as the original input data. In this manner, the autoencoder model can be evaluated, and the model also has a metric to improve.

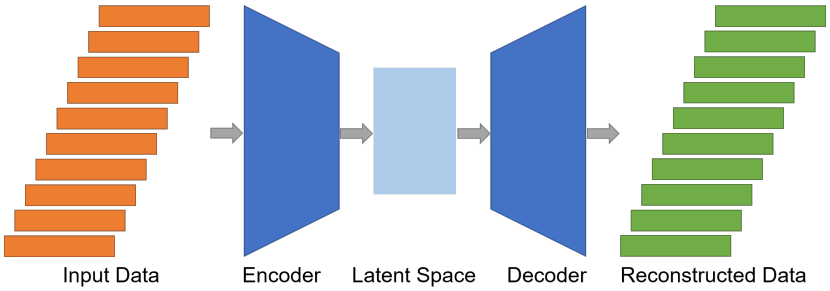


Figure 2.4: The architecture of an autoencoder. Encoder and decoder are composed of neural networks. The encoder compress the input data to encoded code in the latent space, in which the dimensionality is lower than the original data. Afterwards, the decoder reconstruct the input data provided the code from the latent space.

To put it in a mathematical way, the loss function is calculated by comparing the difference between input data and reconstructed data, for example, calculated via mean square error (MSE) eq. 2.1, where X is the input data, Y is the reconstructed data, and n is the number of data points in the input data. When MSE is smaller, meaning the reconstructed data is more similar to the original input data. The optimization can be expressed as eq. 2.2, where θ is the model parameters.

$$MSE(X,\ Y) = \frac{1}{n}, \sum_{i=1}^{n} (X_i - Y_i)^2 \tag{2.1}$$

$$\underset{\theta}{\operatorname{argmin}}\ MSE(X,\ Y). \tag{2.2}$$

2.4 Reinforcement Learning

The concept of reinforcement learning is similar to how people learn. When they behave well, they get rewards. Otherwise, they receive punishments, which can be interpreted as negative rewards. Components of a reinforcement learning problem include state, action, reward, state transition function, and reward function. In each state, a corresponding action is chosen. Afterward, the reward function takes the current state and the action chosen as inputs, and compute corresponding rewards. The state transition function takes also the current state and the action chosen as inputs, and output the next state after taking this action. Fig. 2.5 shows an example of a reinforcement problem.

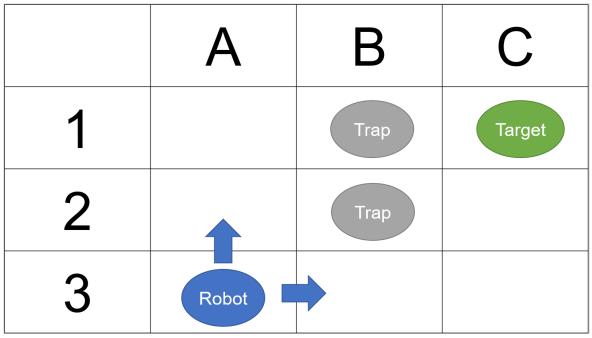


Figure 2.5: An example of a reinforcement problem. The robot can only move up or right, and the goal is to reach the target while avoiding the trap. The robot's position on the map is its state, and the initial state of the robot is A-3. In this case, the action space contains only up and right, and the state space contains the nine possible positions on the map. The reward function should be designed to return a positive value when the state equals C-1, and a negative value when the state is either B-1 or B-2. The state transition function defines what the next state is after certain action taken at a certain state.

Reinforcement learning can be categorized into model-based and model-free learning. The model-based approach learns the state transition function and the reward function, while the model-free approach learns the V-function and the Q-function instead [32]. The V-function stands for the expected return of a certain state, and the Q-function the expected return of a certain action at a state.

3 Variational Autoencoder

In this chapter variational autoencoder (VAE) is introduced. Sec. 3.1 gives the overview of VAE and explains the intuition behind it. The architecture of VAE evolves from autoencoder (AE). The major difference is that the VAE can create artificial data by using the latent space whereas the AE is not able to further utilize the embedded space. Sec. 3.2 explains the mathematical formulation behind the ideas mentioned in the overview section. Sec. 3.3 introduces recurrent neural networks (RNN) and especially long short term memory (LSTM). These are commonly used components in VAE which is used to train on time-series data. Lastly, since VAE is in recent years a popular research topic, variants based on basic VAE are also proposed by researchers. Sec. 3.4 shows two of the variants: beta VAE (β-VAE) and temporal difference VAE (TD-VAE).

3.1 Overview

Autoencoder is an algorithm to reduce the size of a dataset by encoding the data into the latent space, as introduced in Sec. 2.3.2. There is a decoder inside the autoencoder which reconstructs the data given the latent values. However, what would the output data be if the decoder is fed with values not exist in the latent space? Can a decoder be used to generate data by operating in this manner? Fig. 3.1 shows the idea of an ideal data generator. For latent values encoded from the dataset, the generator should be able to reconstruct them. While for values not among the latent values, the generator should output meaningful data. In this case, it generates a shape that has features from both a square and a circle. However, for an autoencoder, it is not guaranteed that any input to the decoder can lead to reasonable outputs.

VAE is designed based on the autoencoder but can generate meaningful new data, and the main difference between the autoencoder and the variational encoder lies in the latent space design. VAE regulates its latent space to be certain distributions, resulting in a meaningful latent space. Fig. 3.2 shows the generated data from a trained generator with a latent space of 2-dimensional Gaussian distribution. The training data are numbers, and the images are generated by latent values on the corresponding locations. We can observe that there are regions for each digit, e.g. the values in the right bottom corner of the latent space result in the number 1. When moving from the right bottom

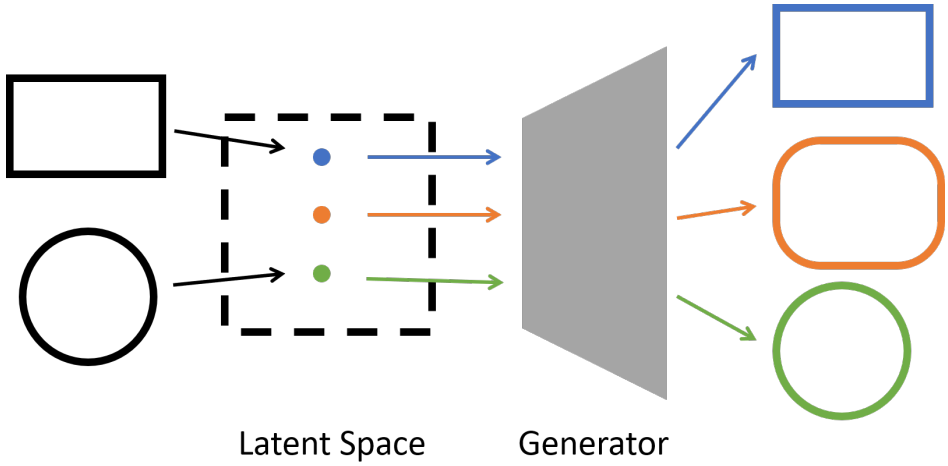


Figure 3.1: A sketch of an ideal data generator. The blue dot represents the code of a square in the latent space, and the green dot stands for the code of the circle. When the code of the square and circle are passed to the generator, a square and a circle should be the output, respectively. The orange dot is a data point between the code of the square and the circle in the latent space. Therefore, when this data point is fed to the generator, the output should also look between a square and a circle.

corner toward the middle, the number 1 transform gradually to the number 6, and then to the number 8. The transitions such as the symbol between 1 and 6 as well as between 6 and 8 are not provided by the training data, but can be generated by a VAE.

VAE achieves this result by regulating the latent space to be a probability distribution, and a Gaussian distribution is often selected. A Gaussian distribution has two parameters, which are the mean and the variance, as shown in eq. 3.1, where the μ stands for the mean and σ^2 the variance. Fig. 3.3 illustrates an example of regulating the latent space to be of a Gaussian distribution. The encoder of the VAE encodes the input data into mean and variance in the latent space, and then randomly samples data from the encoded mean and variance. Ideally, the sampled data are of a Gaussian distribution.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}. \tag{3.1}$$

VAE has two objectives. Firstly, reconstruct the original input data. This is the same idea as the reconstruction loss of an autoencoder. Secondly, the latent space is of a designed distribution. For this purpose, the Kullback–Leibler divergence (KL) loss is used, which is an index showing how similar two distributions are. When the distribution of the latent values is the same as the target distribution, the KL loss equals zero. Fig. 3.4 shows the roles of reconstruction loss and KL loss respectively. When only the reconstruction loss is applied, the differences between different input data are clearly observed in the latent space, but there are many undefined regions and the decoder may not be able

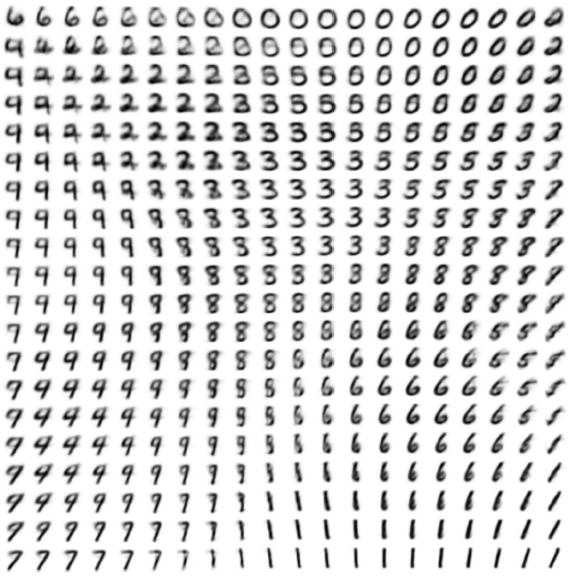


Figure 3.2: An example of generating data using a VAE. Images of numbers are the training data of a VAE with two-dimensional latent space. It shows that a VAE can generate new and meaningful data. [33].

to generate meaningful data from those empty regions. When only KL loss is considered, there are fewer undefined regions in the latent space, but the distinctions between different inputs are unclear. When both losses are implemented together, the benefits of each loss function remain while the disadvantages are overcome. The latent space is of a Gaussian distribution, and the differences among input data are also observable.

Before starting training a VAE, the reparameterization trick should be implemented to make backpropagation functioning. In VAE's architecture, there is a random sampling process before passing the latent values to the decoder, as shown in fig. 3.3. However, the backpropagation cannot compute directly the gradient of a random sampling process. This is when the reparameterization trick is used [33]. The idea is to relocate the sampling process to one additional variable, as depicted in fig. 3.5. The $f(m, v, s)$ in fig. 3.5 is now a function instead of a random sampling process, and the gradients can be computed and propagated through this function as long as it is differentiable.

3.2 Mathematical formulation

Let $\mathbf{x}$ be the input data as well as the ideal reconstructed data and let $\mathbf{z}$ be the latent values. The encoder can be expressed as $p_\theta(\mathbf{z}|\mathbf{x})$ and the decoder $p_\theta(\mathbf{x}|\mathbf{z})$, where p is the true or underlying distribution and θ stands for its parameters.

Eq. 3.2 is the Bayesian equation. We want to estimate the encoding function $p_\theta(\mathbf{z}|\mathbf{x})$. However, it is

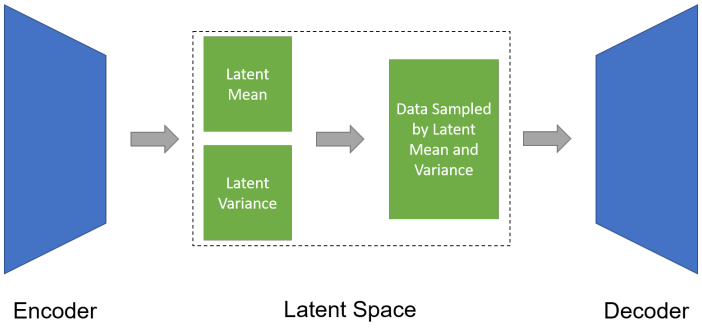


Figure 3.3: How VAE regulates the latent space to be a Gaussian distribution. The main difference between VAE and autoencoder lies between the encoder and the decoder. The encoder of VAE outputs parameters of a Gaussian distribution, the mean and variance. The input data to the decoder are randomly sampled from the latent mean and variance.

not possible to estimate it using the Bayesian equation, because $p(\mathbf{x})$ is intractable.

$$P(\mathbf{z} \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid \mathbf{z}) \cdot P(\mathbf{z})}{P(\mathbf{x})}. \tag{3.2}$$

In order to estimate the encoding function $p_\theta(\mathbf{z}|\mathbf{x})$, we attempt to approximate $p_\theta(\mathbf{z}|\mathbf{x})$ using another distribution $q_\phi(\mathbf{z})$. If $q_\phi(\mathbf{z})$ is very similar to $p_\theta(\mathbf{z}|\mathbf{x})$, we can say $q_\phi(\mathbf{z})$ is a proper approximation of $p_\theta(\mathbf{z}|\mathbf{x})$. Eq. 3.3 is the formula of KL divergence. If $\mathrm{KL}(q_\phi(\mathbf{z})\|p_\theta(\mathbf{z}|\mathbf{x}))$ is minimized, $q_\phi(\mathbf{z})$ is the optimal approximation.

$$\mathrm{KL}(P\|Q) = \sum_{\mathbf{x}\in\mathbf{X}} P(\mathbf{x}) \log\left(\frac{P(\mathbf{x})}{Q(\mathbf{x})}\right). \tag{3.3}$$

Eq. 3.4 attempt to solve the $\mathrm{KL}(q_\phi(\mathbf{z})\|p_\theta(\mathbf{z}|\mathbf{x}))$. The $\mathbb{E}$ in eq. 3.4 represents the expectation of random variables. In continuous case, expectation of a function can be calculated by eq. 3.5. However, the

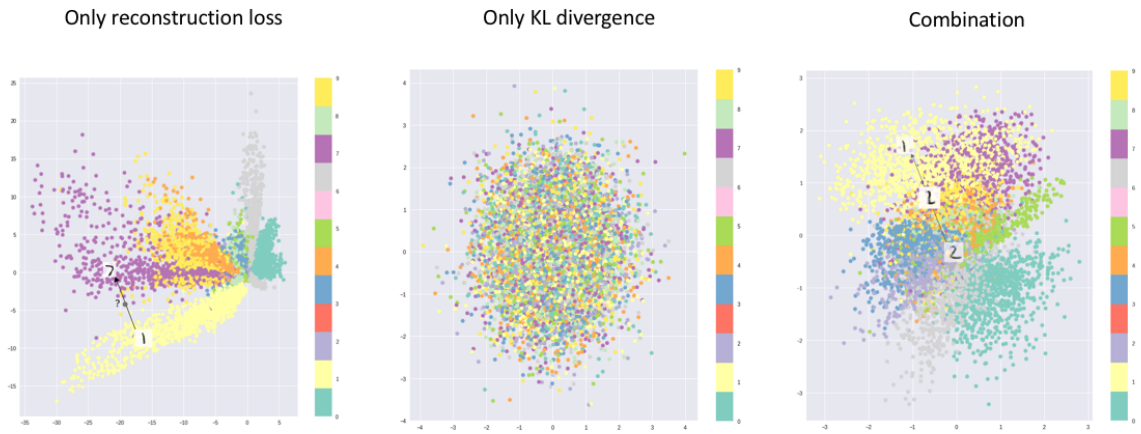


Figure 3.4: Different effects of using only reconstruction loss, only KL loss, and the combination of these two losses. Reconstruction loss focuses on extracting information of the input data, and KL loss enforces the latent space to be of the target distribution. When these two losses are combined, the latent space contains the critical information from input data and the latent values lie in the designed distribution. [34].

calculation is stopped by the term $log\ p_\theta(\mathbf{x})$, because this term is intractable [35].

$$\begin{aligned}
\mathrm{KL}(q_\phi(\mathbf{z})\|p_\theta(\mathbf{z}\mid\mathbf{x})) &= \int q_\phi(\mathbf{z})\log\frac{q_\phi(\mathbf{z})}{p_\theta(\mathbf{z}\mid\mathbf{x})}d\mathbf{z} \\
&= \int q_\phi(\mathbf{z})(\log q_\phi(\mathbf{z}) - \log p_\theta(\mathbf{z}\mid\mathbf{x}))d\mathbf{z} \\
&= \int q_\phi(\mathbf{z})\log q_\phi(\mathbf{z}) - q_\phi(\mathbf{z})\log p_\theta(\mathbf{z}\mid\mathbf{x})d\mathbf{z} \\
&= \int q_\phi(\mathbf{z})\log q_\phi(\mathbf{z})d\mathbf{z} - \int q_\phi(\mathbf{z})\log p_\theta(\mathbf{z}\mid\mathbf{x})d\mathbf{z} \\
&= \mathbb{E}_{q_\phi}[\log q_\phi(\mathbf{z})] - \mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{z}\mid\mathbf{x})] \\
&= \mathbb{E}_{q_\phi}[\log q_\phi(\mathbf{z})] - \mathbb{E}_{q_\phi}\left[\log\frac{p_\theta(\mathbf{x},\mathbf{z})}{p_\theta(\mathbf{x})}\right] \\
&= \mathbb{E}_{q_\phi}[\log q_\phi(\mathbf{z})] - \mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{x},\mathbf{z}) - \log p_\theta(\mathbf{x})] \\
&= \mathbb{E}_{q_\phi}[\log q_\phi(\mathbf{z}) - \log p_\theta(\mathbf{x},\mathbf{z})] + \mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{x})] \\
&= \mathbb{E}_{q_\phi}[\log q_\phi(\mathbf{z}) - \log p_\theta(\mathbf{x},\mathbf{z})] + \log p_\theta(\mathbf{x}).
\end{aligned} \tag{3.4}$$

$$\mathrm{E}[g(X)] = \int_{\mathbb{R}} g(x)f(x)dx \tag{3.5}$$

By rearranging the last equation, eq. 3.6 is obtained.

$$\mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{x},\mathbf{z}) - \log q_\phi(\mathbf{z})] = \log p_\theta(\mathbf{x}) - \mathrm{KL}(q_\phi(\mathbf{z})\|p_\theta(\mathbf{z}\mid\mathbf{x})). \tag{3.6}$$

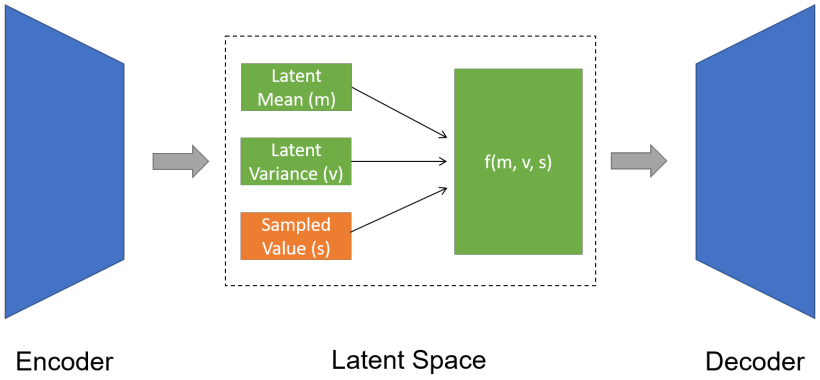


Figure 3.5: This figure shows the idea of the reparameterization trick. Compared with fig. 3.3, the random sampling process is not placed right before decoder anymore, but instead is relocated to one additional latent variable. As a result, the model can propagate gradient computations from the loss back to the parameters.

The second term on the right side is a non-negative term, meaning $\log p_\theta(\mathbf{x})$ is greater or equals to the left side, $\mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z})]$. Therefore, $\mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z})]$ is also named as the evidence lower bound (ELBO). The goal is to minimize $\mathrm{KL}(q_\phi(\mathbf{z})\|p_\theta(\mathbf{z}|\mathbf{x}))$, and when maximizing the ELBO, we are automatically maximizing $\log p_\theta(\mathbf{x})$ and minimizing $\mathrm{KL}(q_\phi(\mathbf{z})\|p_\theta(\mathbf{z}|\mathbf{x}))$. Therefore, the objective function of VAE is maximizing the ELBO. In other words, the loss to be minimized during the training process is the negative ELBO.

The ELBO can be further decomposed as eq. 3.7.

$$
\begin{aligned}
\mathrm{ELBO}(q_\phi) &= \mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z})] \\
&= \mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{z}, \mathbf{x})] - \mathbb{E}_{q_\phi}[\log q_\phi(\mathbf{z})] \\
&= \mathbb{E}_{q_\phi}[\log(p_\theta(\mathbf{x} \mid \mathbf{z})p_\theta(\mathbf{z}))] - \mathbb{E}_{q_\phi}[\log q_\phi(\mathbf{z})] \\
&= \mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{x} \mid \mathbf{z})] + \mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{z})] - \mathbb{E}_{q_\phi}[\log q_\phi(\mathbf{z})] \\
&= \mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{x} \mid \mathbf{z})] + \mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{z}) - \log q_\phi(\mathbf{z})] \\
&= \mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{x} \mid \mathbf{z})] + \int q_\phi(\mathbf{z}) \log \frac{p_\theta(\mathbf{z})}{q_\phi(\mathbf{z})} d\mathbf{z} \\
&= \mathbb{E}_{q_\phi}[\log p_\theta(\mathbf{x} \mid \mathbf{z})] - \mathrm{KL}(q_\phi(\mathbf{z})\|p_\theta(\mathbf{z})).
\end{aligned}
\tag{3.7}
$$

The first term represents the log-likelihood of $\mathbf{x}$ given $\mathbf{z}$. Maximizing this term means minimizing the negative log-likelihood, and can also be interpreted as minimizing the reconstruction loss. The second term can be solved analytically if both $q_\phi(\mathbf{z})$ and $p_\theta(\mathbf{z})$ are of certain distributions. Assuming they are of Gaussian distributions and $q_\phi(x) \sim N(\mu, \sigma)$ and $p_\theta(x) \sim N(0, I)$, a closed-form expression can be obtained, and it is not needed to calculate the ELBO explicitly anymore. This closed-form formula

is derived in eq. 3.8 [36], where J is the number of dimension.

$$
\begin{aligned}
D_{KL}(q_\phi(x)\|p_\theta(x)) &= \mathbb{E}_{z\sim q_\phi(z)}\left[\log\frac{q_\phi(x)}{p_\theta(x)}\right] \\
&= \int \log\frac{q_\phi(x)}{p_\theta(x)} q_\phi(x)dx \\
&= \int (\log q_\phi(x) - \log p_\theta(x)) q_\phi(x)dx \\
&= \int q_\phi(x)\log q_\phi(x)dx - \int q_\phi(x)\log p_\theta(x)dx \\
&= \int N\left(x;\mu,\sigma^2\right)\log N\left(x;\mu,\sigma^2\right)dx - \int N\left(x;\mu,\sigma^2\right)\log N(x;0,I)dx \\
&= \frac{J}{2}\log(2\pi) + \frac{1}{2}\sum_{j=1}^{J}\left(1+\log\sigma_j^2\right) - \frac{J}{2}\log(2\pi) + \frac{1}{2}\sum_{j=1}^{J}\left(\mu_j^2+\sigma_j^2\right) \\
&= -\frac{1}{2}\sum_{j=1}^{J} 1 + \log\sigma_{q,j}^2 - \mu_{q,j}^2 - \sigma_{q,j}^2.
\end{aligned}
\tag{3.8}
$$

3.3 Recurrent Neural Network

When the training dataset is composed of time-series data, RNN layers are often included in the architecture of a VAE. As the name RNN implies, it is designed based on the neural network (NN). In this section, NN is first introduced, and then the RNN, and lastly the LSTM, which is one type of RNN.

3.3.1 Neural Network

Neural network, as the same implies, is a network of neurons. One single neuron is also called perceptron, and perceptron learning is proposed by Rosenblatt [37]. Fig. 3.6 shows the architecture of a perceptron, and the output function is the eq. 3.9. During learning, the perceptron adjusts the weights of inputs to obtain better results.

$$
Y = \sum_{i=0}^{N}(W_i X_i) + Bias. \tag{3.9}
$$

This is the case of a perceptron with single output and single layer. The idea can be extended to multiple perceptrons and also multiple layers in order to create a neural network. Activation functions can also be implemented after each or some of the output layers to add non-linearity to the output functions. Examples of activation functions are sigmoid function (eq. 3.10), hyperbolic tangent function (tanh) (eq. 3.11), rectified linear unit (ReLU) (eq. 3.12), and leaky ReLU (eq. 3.13), where α is a small value. ReLU is a common choice because it takes less computational resources compared with

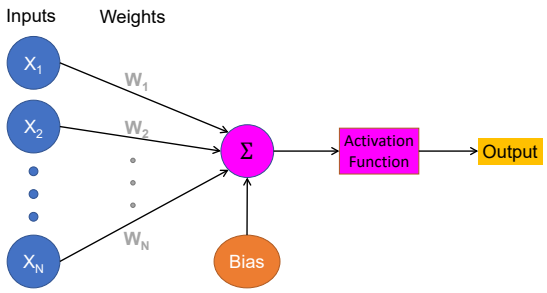


Figure 3.6: This figure shows the architecture of a perceptron. One perceptron generates only one output, but can take many inputs. Each input value is first multiplied with a weight variable, and then added with a bias.

sigmoid and tanh. The drawback of ReLU is the vanishing gradient problem, which is solved by the leaky ReLU by outputting α instead of 0 for cases when the input is not positive.

$$f(x) = \frac{1}{1+e^{-x}}, \tag{3.10}$$

$$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}, \tag{3.11}$$

$$f(x) = \begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases}, \tag{3.12}$$

$$f(x) = \begin{cases} \alpha & x \leq 0 \\ 1 & x > 0 \end{cases}. \tag{3.13}$$

3.3.2 Recurrent Neural Network

RNN is designed for sequential data, where latter data are affected by former data, and is used in many applications e.g. translation between languages, making predictions of time-series data, and speech analysis. The concept of the RNN is to make the neural network ”remember” information from the past. Fig. 3.7 shows different application scenarios of RNN. The input data (red boxes) are first fed to the hidden layer (green boxes), and the outputs (blue boxes) are generated immediately or after collecting a certain amount of input data. The hidden unit takes the current red box and the previous green box as input, and passes the computed information with certain weights to the output layer and to the next hidden unit.

There are, however, two drawbacks of RNN: exploding gradient problem and vanishing gradient problem. During the backpropagation phase, while propagating gradients from the last output all

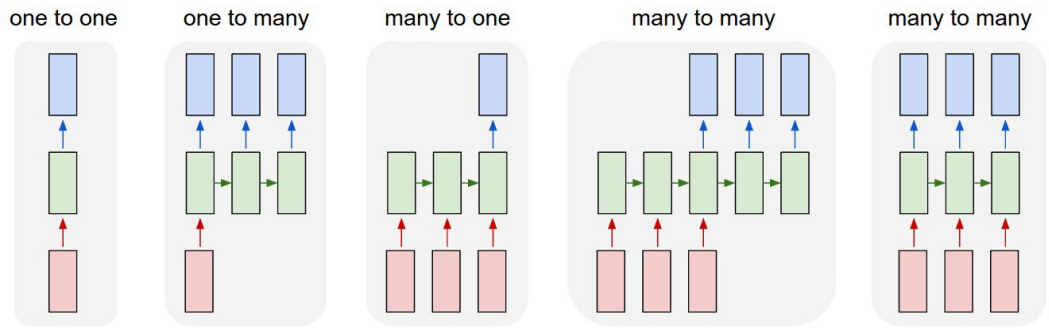


Figure 3.7: Combinations of different input and output types of RNN. The red, green, blue boxes represent input layer, hidden layer, and output layer, respectively. The left-most red box stands for the first input in a data sequence. [38].

the way back to the early hidden unit, the gradient might grow exponentially (exploding gradient problem) or decay exponentially (vanishing gradient problem). This exploding gradient problem can be solved by clipping the gradient to a threshold so that it doesn't grow to a huge number, while the vanishing gradient problem can be solved by the LSTM architecture with suitable activation functions.

3.3.3 Long Short Term Memory

LSTM controls the information flow via three control gates: forget gate, input gate (also known as update gate), and output gate. Fig 3.8 illustrate the architecture of a LSTM unit and how multiple LSTM units are combined together. C, h and x represent cell state, output and input, respectively. The output of the previous LSTM unit and the input of current unit flow first into the forget gate, which controls how much information from the past is allowed to come into the cell state of the current LSTM unit. The forget gate outputs a number which is between 0 and 1, and the previous cell state will later be multiplied by this number. The formula of forget gate is shown in eq. 3.14, where σ is sigmoid function. If f_t is 1 means keeping the previous cell state completely, and 0 means deleting completely. Secondly, the input data for the current LSTM unit arrives at the input gate, which decides how much of the information from the input data is to be integrated into the cell state. Please find the equations of the input gate in eq. 3.15. $\tilde{C}_t$ represent the information input from the current cell, and f_t and i_t controls how the information from the previous cell state and from the current cell state should be blended. C_t is the final cell state of the current cell, which contains information from the past as well as from current input. Lastly, the output gate computes the output based on the cell state and the input data, as shown in eq. 3.16, and o_t represent to which extent of the cell state should be the output. Finally, the output and the cell state are passed to the next LSTM unit.

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \tag{3.14}$$

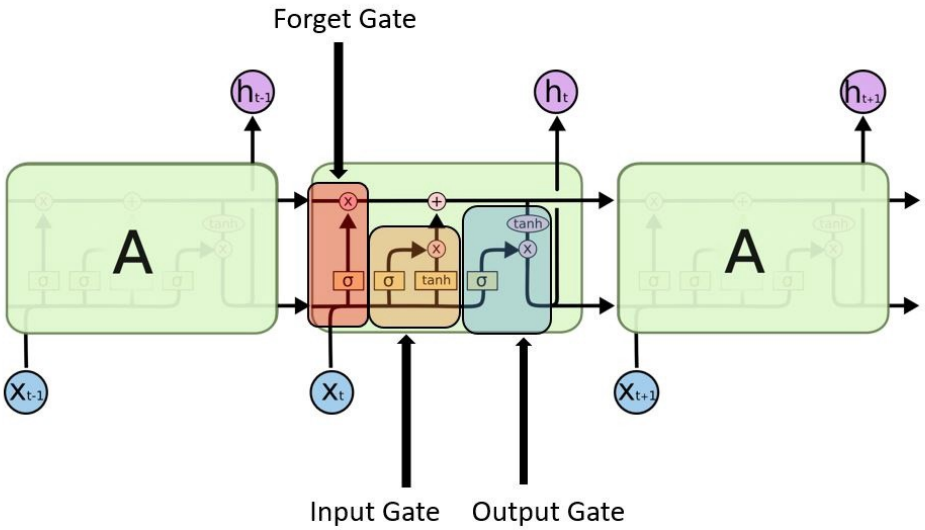


Figure 3.8: Architecture of a LSTM unit. The forget gate controls the weight multiplied to the previous output. The input gate decide how much information to take from the input data. The output gate computes output data of this LSTM unit based on its inner state values. [38].

$$\begin{aligned} i_t &= \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \\ \tilde{C}_t &= \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right) \\ C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \end{aligned} \tag{3.15}$$

$$\begin{aligned} o_t &= \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right) \\ h_t &= o_t * \tanh\left(C_t\right) \end{aligned} \tag{3.16}$$

LSTM overcomes the vanishing gradient problem of RNN by controlling these gates and in general, has better performances than RNN. However, because of these complicated gates, LSTM requires more computational power over basic RNN.

3.4 Other Variants of Variational Autoencoder

VAE has achieved good performances in many different applications, therefore, in recent years, tremendous researches are conducted and variations of design of VAEs are proposed, e.g. VQ-VAE, GMVAE, INFO VAE [39]. Among many variants, self-Consistent VAE, β-VAE and TD-VAE are introduced in this section, and self-Consistent VAE is further implemented and discussed in the following chapters.

3.4.1 Self Consistent Variational Autoencoder

Co-Reyes et al. proposed an algorithm that combined VAE and reinforcement learning (RL) [40]. The self-consistent VAE is trained so that the latent can represent trajectories, based on which the RL models can be trained. This VAE contains one encoder and two decoders. The encoder of this self-consistent VAE takes sequences of states as input, and one of the decoder, the state decoder, generates state sequences. The other decoder, the policy decoder, generates policies for RL models to achieve such state sequences. Details of the algorithm can be found in fig. 3.9. The state information are looped within the encoder, and then the latent mean and latent variance are calculated. Samples are drawn from the latent mean and variance, and are fed to both decoders. In the state decoder, the first state is given, and all following states are decoded by taking the latent samples and the previously decoded state as input. While for the policy decoder, it focuses on generating the actions required for going from current state to the next state.

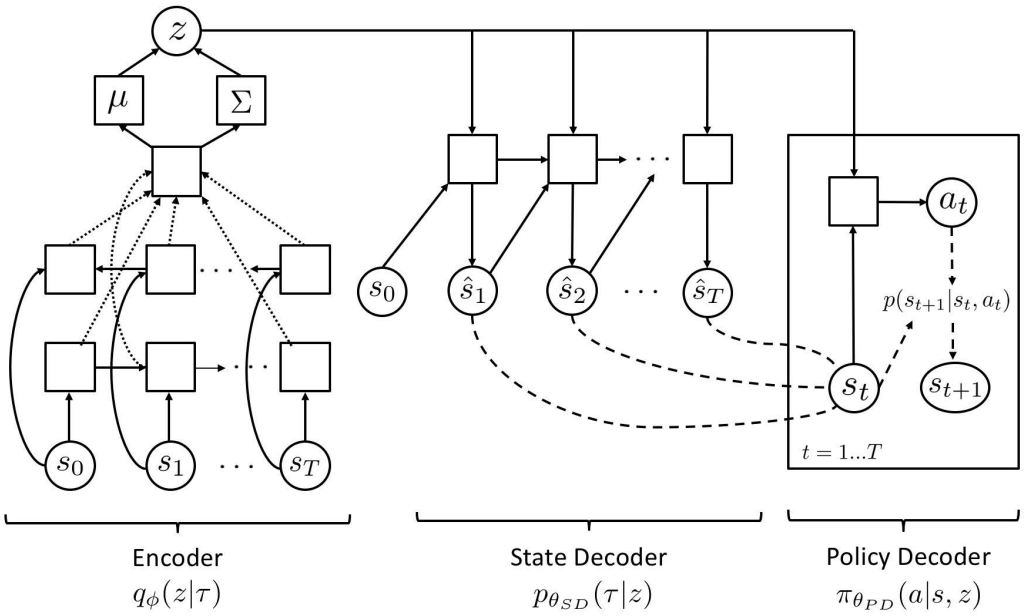


Figure 3.9: [40].

3.4.2 Beta Variational Autoencoder

The concept and the implementation of β-VAE is similar to the original VAE. Recall that the loss function of a VAE consists of two losses, the reconstruction loss and the KL loss. The idea of the β-VAE is to add a weighting hyper parameter β to the KL loss. When $\beta = 1$, it is the same as VAE. When $\beta > 1$, the model focuses more on reducing the KL loss. If the model can decrease KL loss for a certain percentage, this reduction is magnified by β when calculating the total loss, this is the reason why the model has more motivation to emphasize on reducing KL loss, resulting in better latent space

encoding.

The latent space of a β-VAE can be more disentangled than the one of VAE when $\beta > 1$. Higgins et al. defines a disentangled representation as "Single latent units are sensitive to changes in single generative factors, while being relatively invariant to changes in other factors" [41]. For example, if a dataset contains only images of squares with different sizes and different brightness, and if its latent space is two dimensional and disentangled, the two dimensions may represent the size and the brightness of a square, respectively. With a disentangled latent representation, the generator is more capable of generating specific output, e.g. an image of a square with a small size and low brightness.

3.4.3 Temporal Difference Variational Autoencoder

TD-VAE is a very recent variation of VAE, proposed in 2019 by Gregor et al [42]. For most models working with time-series data, the predictions of states of the current timestamp is calculated based on information of the previous timestamp. However, TD-VAE aims to make predictions directly from a certain timestamp in the past instead of the previous timestamp. In other words, instead of predicting only the next second, but predicting directly several steps ahead in the future. This is where the name "temporal difference" comes from. The advantage is that, when predicting into several timestamps into the future, it doesn't need to calculate through all the timestamps in between.

4 Methodology

The goal of this thesis is to reduce the dimension of the given dataset with unsupervised learning model. The biggest challenge of this dataset is that the lengths of each data sequence are different. The investigated machine and its dataset are introduced in sec. 4.1. Secondly, the windows method used to overcome the challenge of variable sequence length is explained in sec. 4.2. The last two sections illustrate the architecture and hyper parameters of the unsupervised models that are implemented: sec. 4.3 for autoencoder (AE) models, and sec. 4.4 for variational autoencoder (VAE) models.

4.1 Dataset

This research focuses on the simulated data of a pneumatic machine, whose scheme is shown in fig. 4.1. The cylinder is actuated by the switching valve and moves the load mass horizontally. X is the displacement of the load mass. The throttle L_{intern}, L_{ext_rod}, and L_{ext_piston} are adjustable and control the internal leakage, external leakage at the rod's side, and external leakage at the piston's side in simulations. Cases with and without leakage are both simulated, and multiple categories of leakage situation do not occur together in a same test, In each experiment, the cylinder velocity and the load mass are set, and the actuating valve status and the cylinder displacement are recorded as time-series data. The sampling period is 1 millisecond, and the velocity can be calculated by the difference between previous and next displacement divided by the time elapse.

Fig. 4.2 illustrate the data sequence of one experiment. Valve status data is binary, 1 meaning open and 0 meaning closed. When the valve opens, the cylinder reaches outward, therefore the displacement increases. When the valve closes, the cylinder retracts, and the displacement decreases until it is back to the starting point. With different configuration of velocity and load mass, the displacement profile differs, and the time required for the cylinder to complete a cycle also varies, resulting in data sequences with different lengths. The histogram of time duration of all experiments can be found in fig. 4.3.

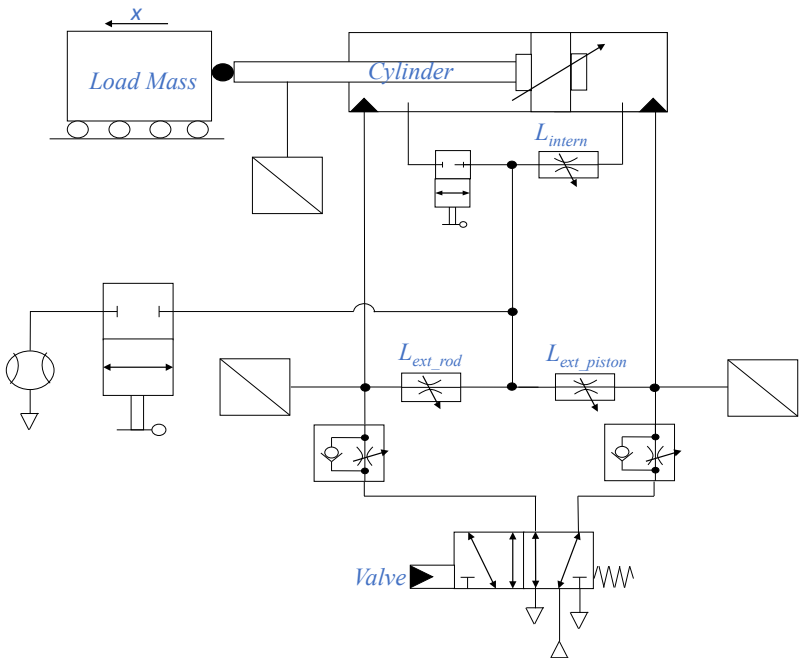


Figure 4.1: Scheme of the interested pneumatic machine. The cylinder is driven to move the load mass horizontally, and the data of the cylinder displacement x and the status of actuating valve are stored.

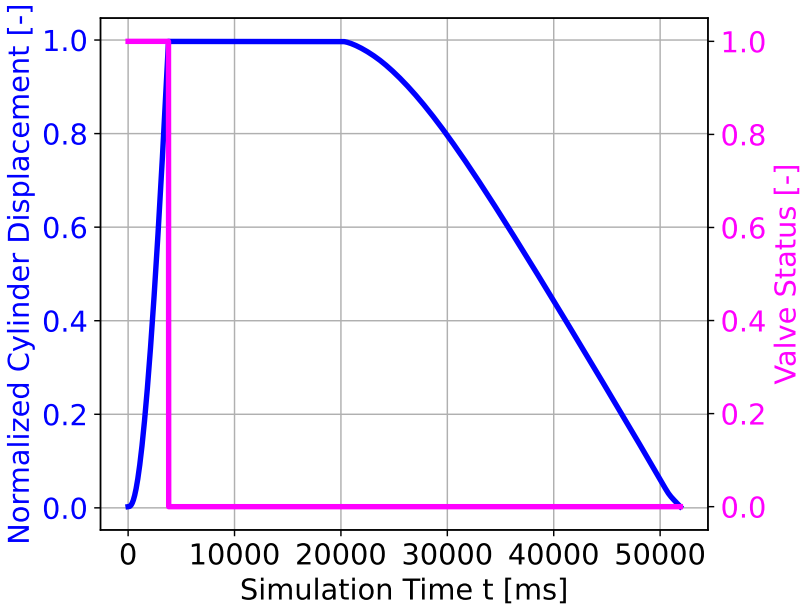


Figure 4.2: An illustration of the time-series data of the pneumatic machine. The valve opens (Valve Status $= 1$), the cylinder displacement rises. The valve closes (Valve Status $= 0$), the value of cylinder displacement falls back.

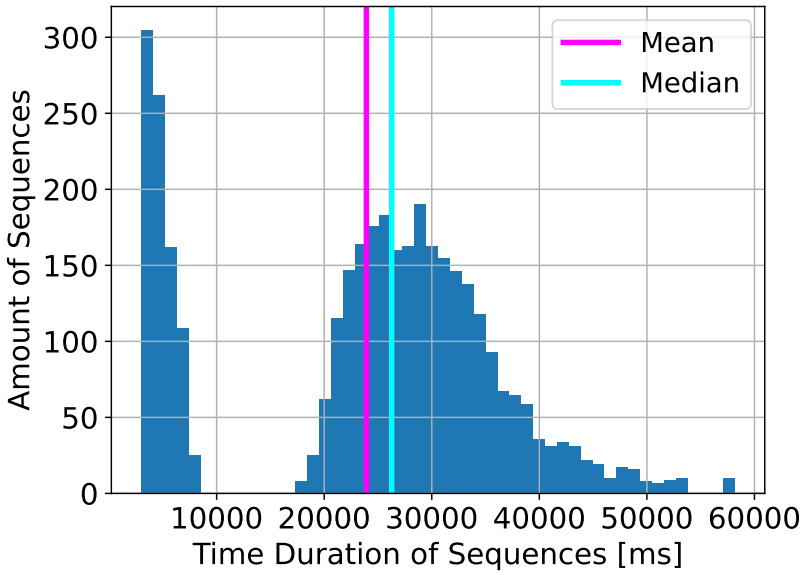


Figure 4.3: Histogram of time duration for each data sequence. Due to different operation settings, the time of each simulated experiment can be greatly different among one another.

4.2 Analyzing by Windows

Variable sequence length is problematic when building models, because the architecture of most machine learning models require the input data to be of the same length. A common way of modelling data sequences with variable length is to force them into the same length by either adding artificial data points to shorter sequences or deleting data points from longer sequences, or both. However, as shown in fig. 4.3, the lengths of sequences in the dataset used in this research vary too much. If all sequences are shaped into the mean or median length, more than 50% of the data points in the longest sequence will be deleted, and the amount of artificial data appended to the shortest sequences will be more than 100% of the amount of its original data. With such huge changes to the dataset, it is risky to claim that the newly generated dataset represents the same information as the original one. Therefore, this research attempt to do the least changes to the dataset as possible by cutting each sequence into segments of the same size, which is smaller than the shortest data sequence. In the following, such segments are called ”windows”. The conversion rule of data sequences to windows is depicted in fig. 4.4 and the corresponding algorithm is stated in algo. 1.

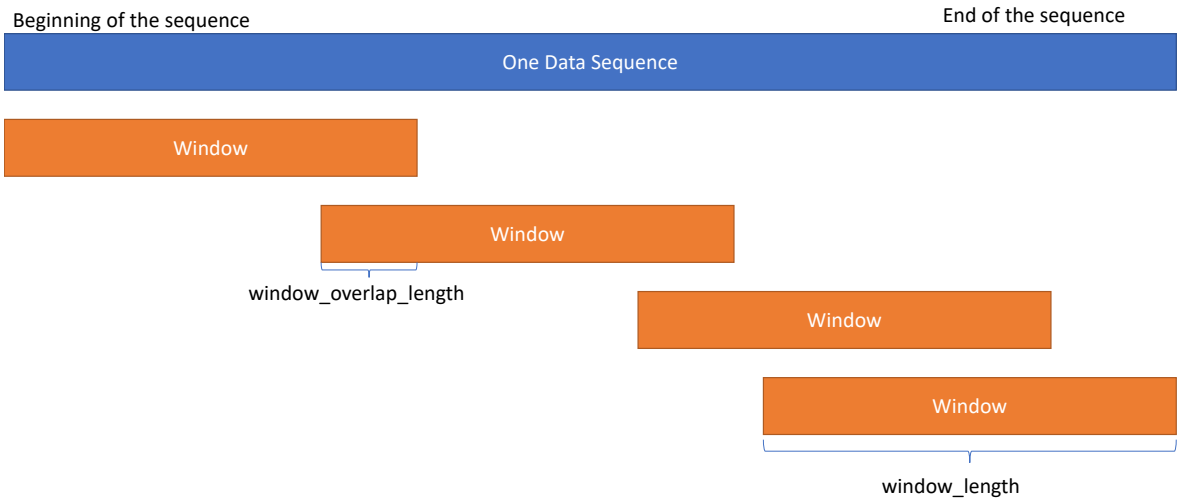


Figure 4.4: Each sequence is sliced to multiple windows with the fixed window length, and each window has a fixed overlap with its neighboring windows except for the last window.

Algorithm 1 Slice a sequence to multiple windows

1: Given: A data sequence, $window\ length\ =\ 200$ and $window\ overlap\ length\ =\ 20$
2: $stored\ windows$ = empty array, will be used to store all the resulting windows
3: Append the first $window\ length$ data points to $stored\ windows$
4: $last\ index$ = index of the last data point of the first window
5: **while** $last\ index <$ the length of this input data sequence **do**
6: $\quad begin\ index = last\ index - window\ overlap\ length$
7: $\quad last\ index = begin\ index + window\ length$
8: $\quad newly\ sliced\ window\ =$ data points from the $begin\ index$ to the $last\ index$ of this data sequence
9: $\quad$ Append $newly\ sliced\ window$ to $stored\ windows$
10: **end while**
11: Append the last $window\ length$ data points to $stored\ windows$

For instance, if the sequence has 700 data points and the window length and the window overlap length are set to be 200 and 20 respectively, then the first window contains the first 200 data points of this sequence, the second window the $180th$ data point until the $380th$, the third window the $360th$ until the $560th$, the forth also the last window contains consequently the last 200 data points.

4.3 Autoencoder Models

This thesis use the AE algorithm to reduce the dimension of the data sequences and have implemented 12 different AE models by using Python and the Pytorch framework. These 12 different models share a general architecture which is illustrated in fig. 4.5.

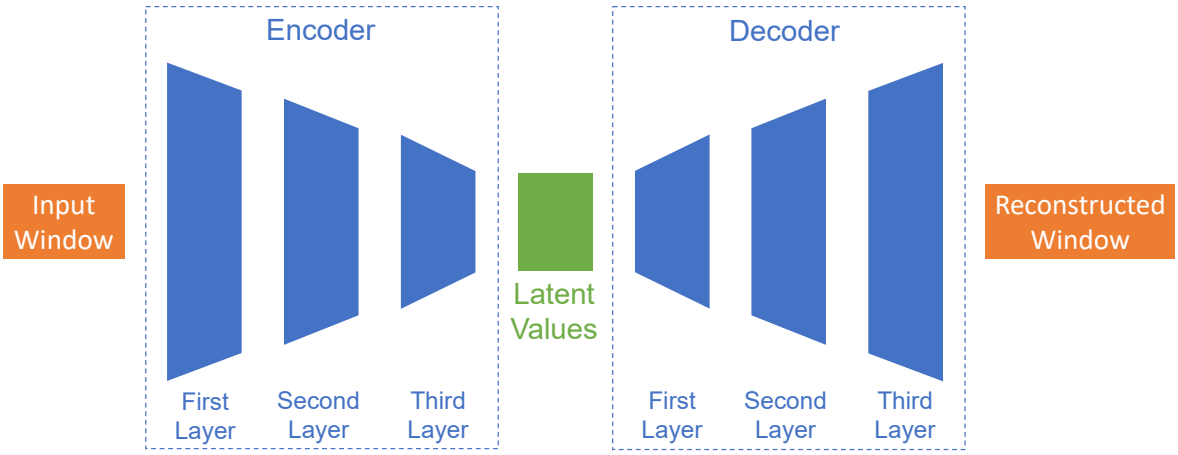


Figure 4.5: Each window is compressed via three layers in the encoder into latent values with lower dimensions. A window can be reconstructed via passing through three layers in the decoder.

The structures of the encoder and the decoder are symmetrical. The second and third layer of the encoder as well as the first and second layer of the decoder are linear neural networks. For half of the models, the first layer of the encoder and the third layer of the decoder are linear neural networks as well, but the other 6 models use LSTM structure in these two layers. The training dataset of all models are 1200 data sequences. Every data sequence is cut into windows and then shuffled. 80% of the windows become the training dataset and the rest 20% become the testing dataset. For half of the models, the training data contain three features, namely the displacement of the cylinder, valve status, and the velocity calculated from the displacement, while the other 6 models don't contain the velocity feature. The latent size of one-third of the models is set to be the same as the number of features, namely 2 or 3, and for the other one-third of the models the latent sizes is set to 4, and the rest set to 64. These variations result in the 12 different AE models as listed in table 4.1.

Model Name	LSTM	Feature Size (F)	Latent Size (L)
Linear_F2_L2	No	2	2
Linear_F2_L4	No	2	4
Linear_F2_L64	No	2	64
Linear_F3_L3	No	3	3
Linear_F3_L4	No	3	4
Linear_F3_L64	No	3	64
LSTM_F2_L2	Yes	2	2
LSTM_F2_L4	Yes	2	4
LSTM_F2_L64	Yes	2	64
LSTM_F3_L3	Yes	3	3
LSTM_F3_L4	Yes	3	4
LSTM_F3_L64	Yes	3	64

Table 4.1: Table of the 12 AE models being analyzed and their hyperparameters.

The process of training a model consist of 151 epochs. The training dataset is divided into mini-batches with batch size $= 25$. In each epoch, the model is trained with the training dataset. After feeding each mini-batch to the model, the loss is calculated and backpropagation is performed to update the parameters of the model. All losses of each mini-batch are summed together and called the training loss. After training the model with all mini-batches, the testing dataset which have not been ”seen” by the model are fed to the model as a means of evaluating the model. The loss is calculated and called the testing loss, and then a new epoch starts. This process is shown in algo. 2.

After the AE models are established, classification models are trained from the latent values generated by the AE models, as depicted in fig. 4.6. Compared with the original AE structure (fig. 2.4, the latent values are not fed to the decoder anymore, but are used to train classification models, and to eventually predict the status of leakage. Leakage categories are, as explained in sec. 4.1, no leakage, external rod leakage, external piston leakage, and internal leakage.

Algorithm 2 Process of training a model

Given: 1200 data sequences with different lengths
Slice each data sequence into multiple windows and store all windows
Shuffle all windows
Let 80% of all windows become the training dataset and 20% the testing dataset
The training dataset is divided into mini-batches with batch size $= 25$
for $epoch = 0, 1, 2, \ldots, 150$ **do**
    for each mini-batch **do**
        Feed the mini-batch into the encoder and output the latent values
        Feed the latent values into the decoder and output the reconstructed mini-batch
        $loss\ of\ each\ batch =$ Mean Square Error between the reconstructed mini-batch with the original mini-batch
        Backpropagate the loss back to each parameter and update it accordingly
    end for
    $training\ loss =$ summation of all the $loss\ of\ each\ batch$
    Feed the testing dataset into the encoder and result in the latent values
    Feed the latent values into the decoder and result in the reconstructed testing dataset
    $testingloss =$ Mean Square Error between the reconstructed testing dataset with the original testing dataset
end for

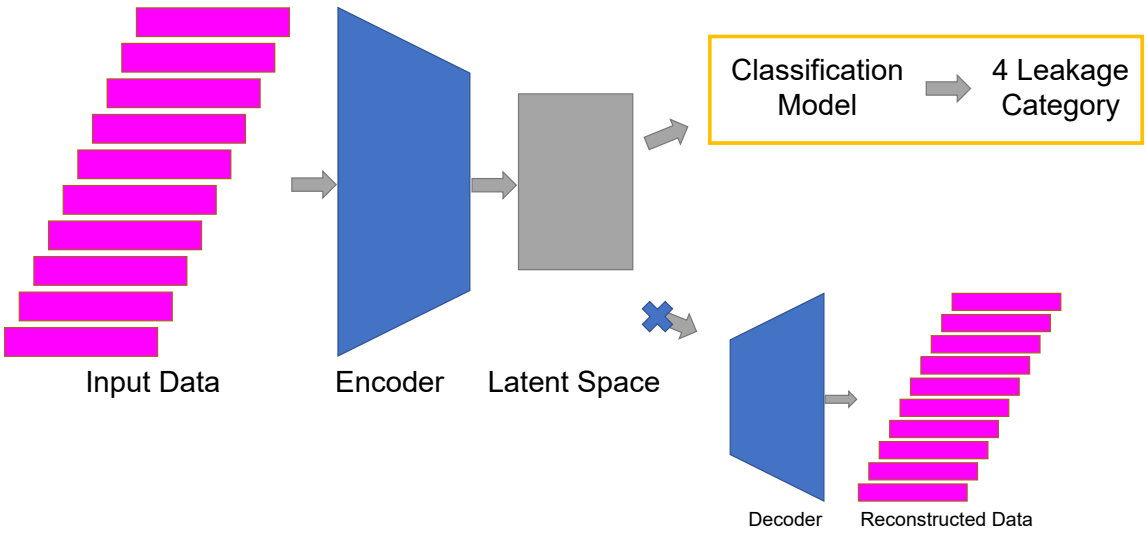


Figure 4.6: Each window is compressed via three layers in the encoder into latent values with lower dimensions. A window can be reconstructed via passing through three layers in the decoder.

Ideally, the classification task is to take the latent values from a data sequence as input, and then predict the leakage category of this data sequence. However, the AE models takes windows as input instead of sequences. When a sequence is longer, it is sliced into more windows than a shorter sequence, and because each window leads to one latent value, a longer sequence results in more latent values in total than a shorter sequence. It is problematic to train a classifier with different sizes of input data.

There are two approaches to solve this problem: One is to make the output of each sequence become the same size and train a classifier that classify sequences, and the other idea is simply to train a classifier that classify windows instead of sequences. The first approach is shown in algo. 3. The idea is to calculate the mean value of all latent values resulted from all windows of one data sequences. Therefore, one data sequence results in only one output, and the input data to the classifier are of the same size. The other approach is explained in algo. 4. In this approach the concept of sequences is taken away and each window is treated individually. The latent value resulted from a window is fed into the classifier and the classifier tells the leakage category of this window.

Algorithm 3 Classifying per Sequence

```
Given: 3520 data sequences and 12 AE models
for each AE model do
    for each data sequence do
        Slice each sequence into windows
        Feed the windows into this AE model and obtain latent value per window
        Compute mean of all latent values to be the output of this data sequence
    end for
    Obtain a dataset that contains 3520 data points
    80% of the dataset become the training dataset for the classifier and 20% the testing dataset
    Train the classifier with the training dataset (n_estimators = 100)
    Evaluate the classifier with the testing dataset
end for
```

As for the structure of a classifier model, this research implemented both random forest tree (RF) and multi-layer perceptron (MLP) classification models. Eventually, two approaches and two model structures are applied to latent values resulted from 12 AE models, there are $2 * 2 * 12 = 48$ classifier models built, as shown in table 4.2.

Algorithm 4 Classifying per Window

```
Given: 3520 data sequences and 12 AE models
for each AE model do
    for each data sequence do
        Slice each sequence into windows
        Attach the leakage category of this sequence to all the windows
        Feed the windows into this AE model and obtain latent value per window
    end for
    Obtain a dataset that contains all latent values of all windows which has 469159 data points
    (there are 469159 windows in total)
    80% of the dataset become the training dataset for the classifier and 20% the testing dataset
    Train the classifier with the training dataset (n_estimators = 100)
    Evaluate the classifier with the testing dataset
end for
```

	Random Forest	Multi-Layer Perceptron
Classification per Sequence	12	12
Classification per Window	12	12

Table 4.2: Components of the 48 classification models. There are 12 AE models and therefore generate 12 sets of latent values. 12 classifiers are based on RF model and the input data of classifier are data sequences. Another 12 RF classifiers take sliced windows as input data. The other 24 classifiers are all based on MLP and are also equally divided to taking data sequences as input versus taking windows as input.

4.4 Variational Autoencoder Models

In addition to AE models, two different VAE models are also implemented in this research. The first one is based on the basic structure shown in fig. 3.3. This model consists of an encoder and a decoder. The architecture of the encoder and decoder are shown in fig. 4.7 and fig. 4.8, respectively. The input data to the encoder are the windows of all input sequences. These windows are first compressed by one LSTM layer and two linear layer, and then further compressed via two separate linear layers to give latent mean and latent variance. Before proceeding in the decoder, latent samples are first generated by random sampling from the latent mean and latent variance. Afterward, the latent samples are expanded through two linear layers and two LSTM layers, and the reconstructed windows are yielded. The reconstructed sequences can be calculated by putting the reconstructed windows together. The detailed algorithm is mostly the same as algo. 2 except what is done in the 150 training epochs. The

different procedures of training this VAE model in the training epochs are shown in algo. 5.

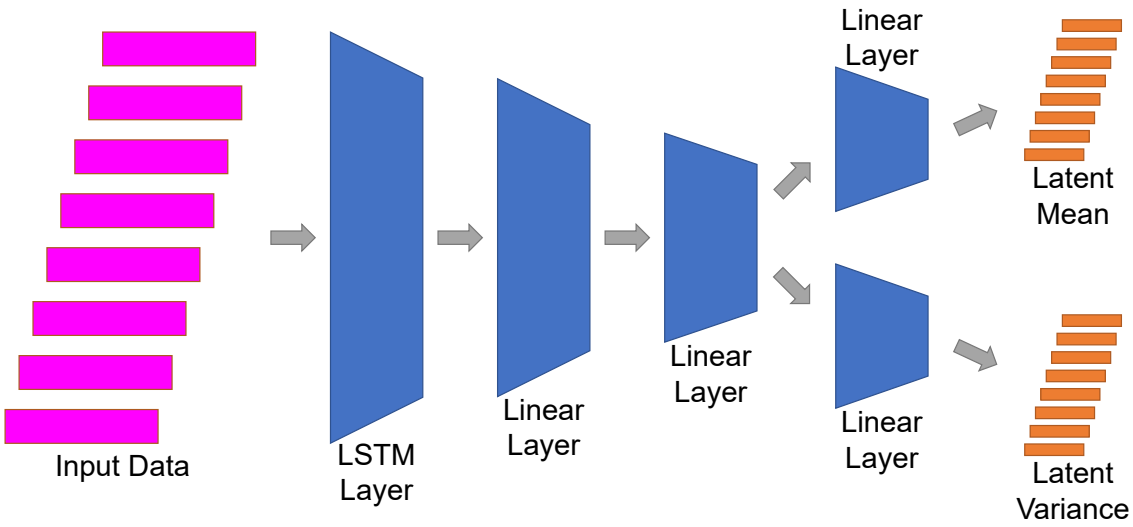


Figure 4.7: The encoder architecture of the basic VAE model in this research. Input data are encoded to latent mean and latent variance via an encoder which has one LSTM layer and three linear layers.

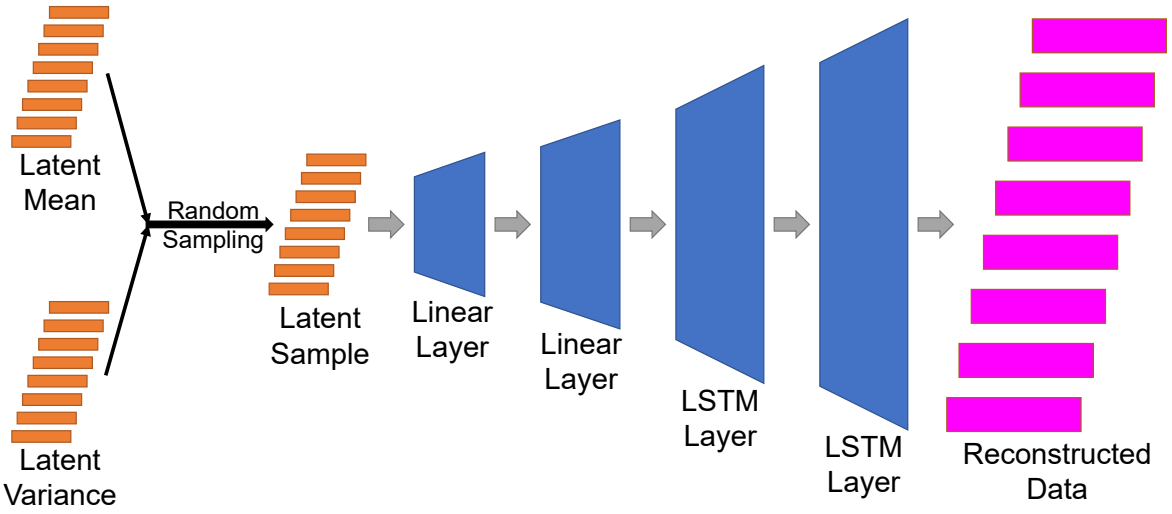


Figure 4.8: The decoder architecture of the basic VAE model in this research. Latent samples are generated by random sampling from the latent mean and variance resulted from the encoder. Latent samples are decoded to reconstruct the original data via two linear layers and two LSTM layers.

Algorithm 5 VAE model building process within the training loop

for $epoch = 0, 1, 2, \ldots, 150$ **do**
  for each mini-batch **do**
    Feed the mini-batch into the encoder and result in the latent mean and latent variance
    Use the mean and variance to perform random sampling and generate latent samples
    Feed the latent samples into the decoder and result in the reconstructed mini-batch
    $recon\ loss\ of\ each\ batch =$ Mean Square Error between the reconstructed mini-batch with the original mini-batch
    $KL\ loss\ of\ each\ batch =$ KL divergence between the latent space and standard normal distribution
    $total\ loss\ of\ each\ batch = recon\ loss\ of\ each\ batch + KL\ loss\ of\ each\ batch$
    Backpropagate the $total\ loss\ of\ each\ batch$ back to each parameter and update it accordingly
  end for
end for

The other VAE model is inspired by the self-consistent structure as described in sec. 3.4.1. This model also consists of an encoder and a decoder. Data sequences are first sliced into windows, and windows from the same sequence are grouped together. The encoder, as depicted in fig. 4.9, takes the grouped windows as input, and feed the windows sequentially to its first layer, a LSTM layer. After processing the first window, the LSTM layer update its weights, and the updated weights are used when taking the next window in. After the LSTM has processed all the windows, the final output of this layer is passed to the following layers, and the data size is further reduced. Finally, one data sequence results in two data points: latent mean and latent variance.

The input data to the decoder (fig. 4.10) is again the data points sampled using the latent mean and latent variance, and one data point represent one data sequence. To reconstruct a sequence, the latent sample is first decoded via two linear layers and one LSTM layer, and the last LSTM layer reconstructs the windows sequentially. The first window is directly given by the original data. The second window is reconstructed by taking the first window and the decoded latent as input to the last layer. The reconstructed second window together with the decoded latent are used to reconstruct the third window, and every window is used to reconstruct the next window. In the end, all windows resulted from the decoder form the data sequence. The detailed algorithm is shown in algo. 6.

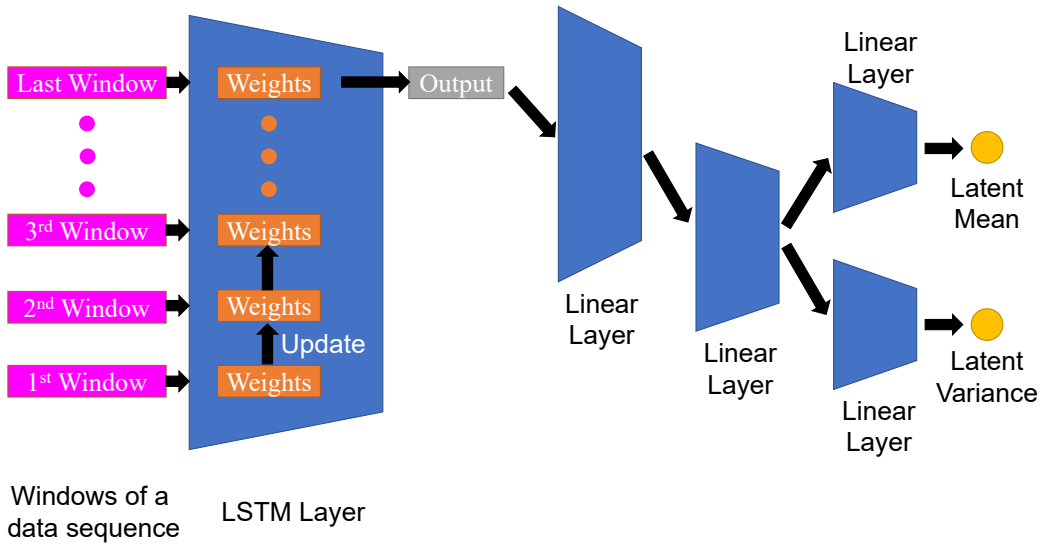


Figure 4.9: The encoder architecture of the self-consistent VAE model in this research. For each data sequence, the first layer of encoder loops through all the windows sequentially to give an output. The output is further compressed via three linear layers and become one data point for each sequence.

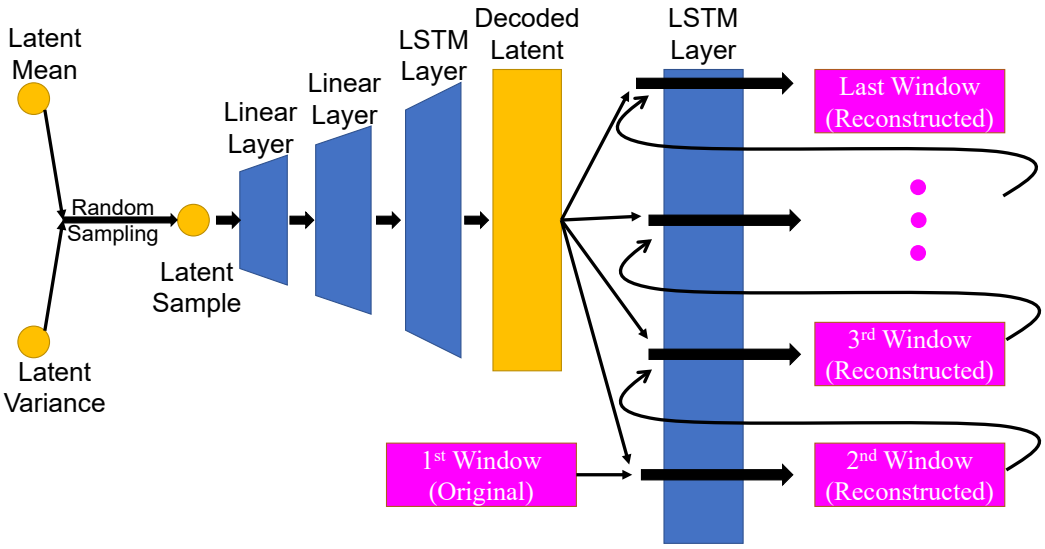


Figure 4.10: The decoder architecture of the self-consistent VAE model in this research. From each latent mean and variance, one latent sample is generated, and is decoded via two linear layers and one LSTM layer. The decoded latent together with the first window of the original data reconstruct the windows of a data sequence.

Algorithm 6 Detailed self-VAE model building process within the training loop

Given: 1200 data sequences with different lengths
Let 80% of data sequences become the training dataset and 20% the testing dataset
The training dataset is divided into mini-batches with batch size $= 25$
for $epoch = 0, 1, 2, \ldots, 150$ **do**
    for each mini-batch **do**
        for each data sequence **do**
            Slice the sequence into multiple windows
            for each window **do**
                This window is fed to the first layer of the encoder
                The weights inside this layer are updated
            end for
            $encoder\ first\ output =$ output of the first layer of the encoder
            $encoder\ second\ output = encoder\ first\ output$ further compressed by the two linear layers of the encoder
            $latent\ mean = encoder\ second\ output$ encoded by a linear layer
            $latent\ variance = encoder\ second\ output$ encoded by a different linear layer
            $latent\ sample =$ one data point sampled by $latent\ mean$ and $latent\ variance$
            $decoded\ latent = latent\ sample$ decoded by two linear layers and one LSTM layer
            $next\ reconstructed\ window =$ first window of the original data
            $all\ reconstructed\ windows =$ a list contains $next\ reconstructed\ window$
            for $i = 2, \ldots, number\ of\ total\ windows\ of\ this\ sequence$ **do**
                Input $decoded\ latent$ and $next\ reconstructed\ window$ to last layer of decoder
                $decoder\ output =$ decoder's output
                $decoder\ output$ is appended to $all\ reconstructed\ windows$
                $next\ reconstructed\ window = decoder\ output$
            end for
        end for
        $recon\ loss\ of\ each\ batch =$ Mean Square Error between the reconstructed mini-batch with the original mini-batch
        $KL\ loss\ of\ each\ batch =$ KL divergence between the latent space and standard normal distribution
        $total\ loss\ of\ each\ batch = recon\ loss\ of\ each\ batch + KL\ loss\ of\ each\ batch$
        Backpropagate the $total\ loss\ of\ each\ batch$ back to each parameter and update it accordingly
    end for
end for

5 Result

In this study, every data sequence is cut into smaller windows, and then they are compressed via different models into one latent value which is of 2, 3, 4, or 64 dimensions. In order to evaluate if a model is suitable for this task, the losses calculated during the training phase of the model are investigated, and the data reconstructed from the latent values are compared with the original data sequence. In addition, the lower-dimension latent values are also used for a classification task, in order to show that this study can have positive influences on practical applications.

5.1 Autoencoder

In this section, results regarding the 12 Autoencoder models mentioned in sec. 4.3 are presented. Sec. 5.1.1 presents the loss information during the training phase, which can be used to analyze whether a model is able to learn to perform this task or not, as well as detecting the overfitting and underfitting problem. Sec. 5.1.2 lists the reconstruction loss and compares the original data and the reconstructed data. It can be interpreted from the reconstructed data whether the reduced latent values possess the crucial information of the input data or not. Sec. 5.1.3 shows the classification result of using the latent values. The classification results are important for us to show that this study can also be used to solve practical unsupervised machine learning problems.

5.1.1 Loss Result

12 Autoencoder models are trained, each trained with 151 epochs. Table 5.1 shows the averaged training time per epoch of each model. The time are in minutes and are rounded to the first digit. The relationship between training time and parameter settings can be better seen from fig. 5.1.

Feature Size	Latent Size	Linear Model [Minutes]	LSTM Model [Minutes]
2	2	0.3	10.0
2	4	0.3	10.3
2	64	1.4	12.1
3	3	0.3	9.4
3	4	0.3	9.9
3	64	1.4	11.9

Table 5.1: The averaged time required in minutes for one epoch in a training process.

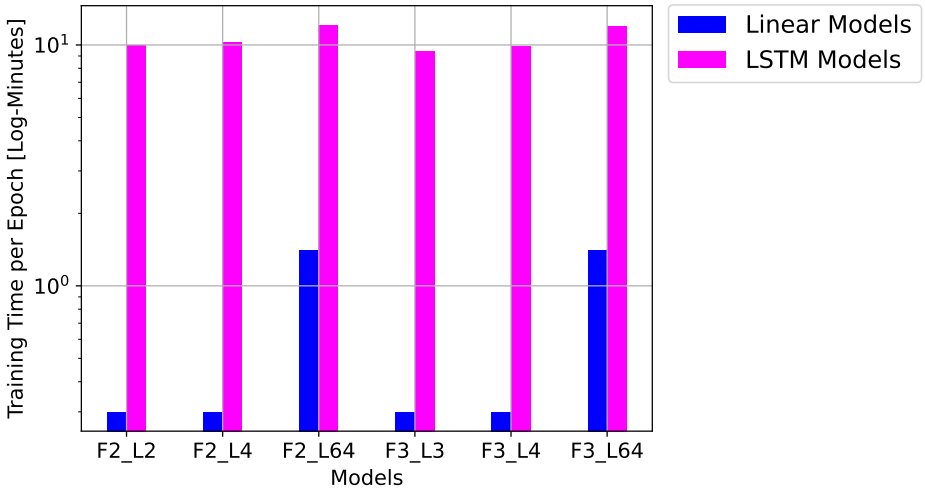


Figure 5.1: Averaged training time of a epoch of each model.

The logarithm values of training loss and the testing loss during the model building process are logged and plotted in this section from fig. 5.2 to fig. 5.13. Each figure contains the training and testing loss during the 151 epochs when building a model. The meaning of the model name please refer to the model list in table 4.1.

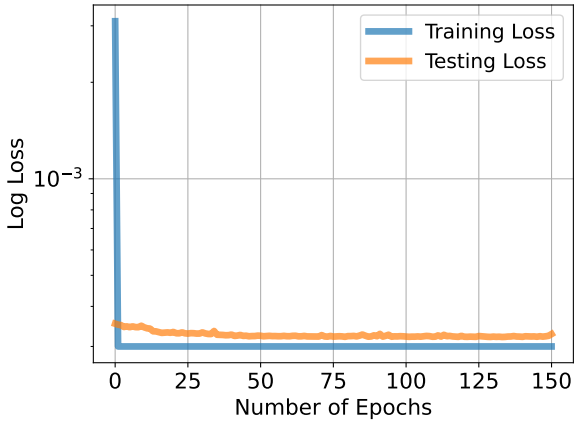


Figure 5.2: Log Loss of Linear_F2_L2

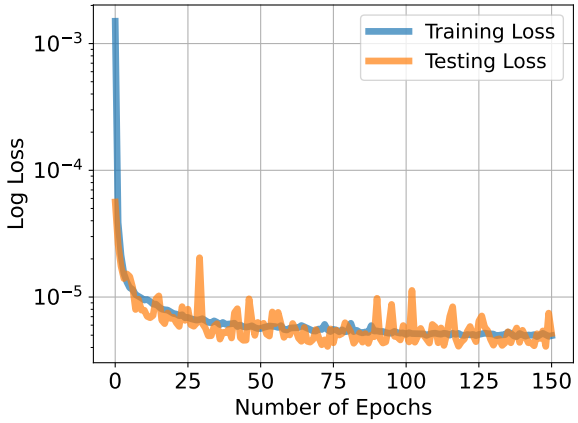


Figure 5.3: Log Loss of LSTM_F2_L2

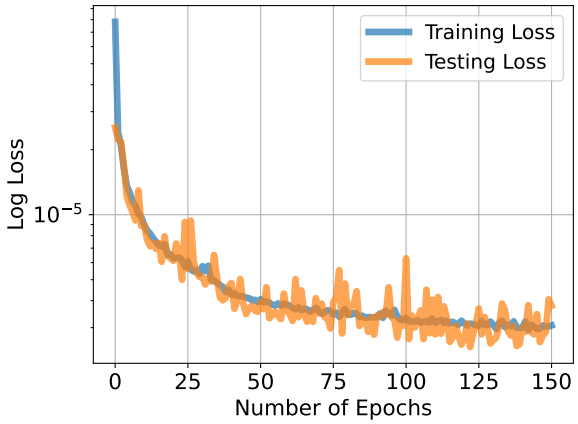


Figure 5.4: Log Loss of Linear_F2_L4

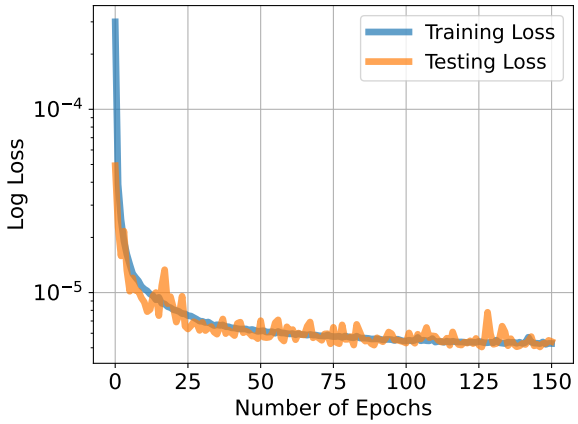


Figure 5.5: Log Loss of LSTM_F2_L4

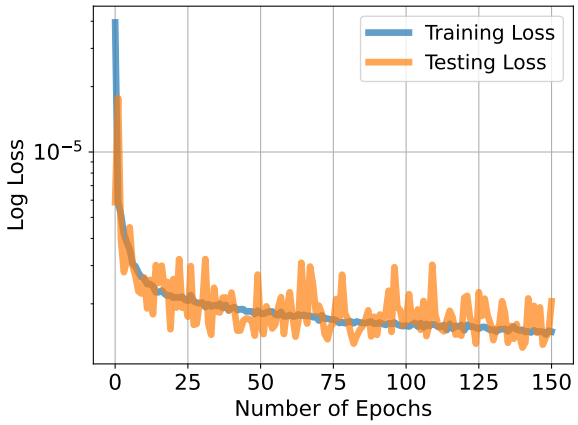


Figure 5.6: Log Loss of Linear_F2_L64

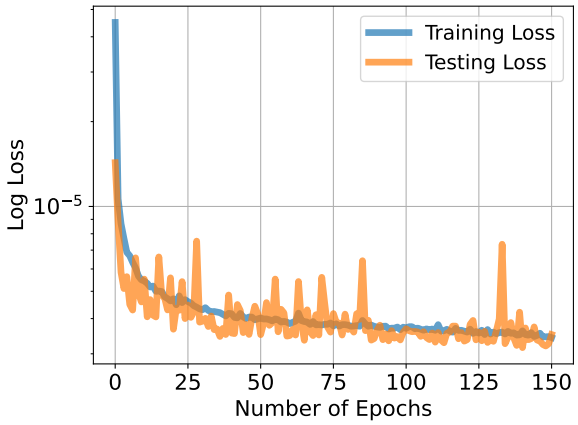


Figure 5.7: Log Loss of LSTM_F2_L64

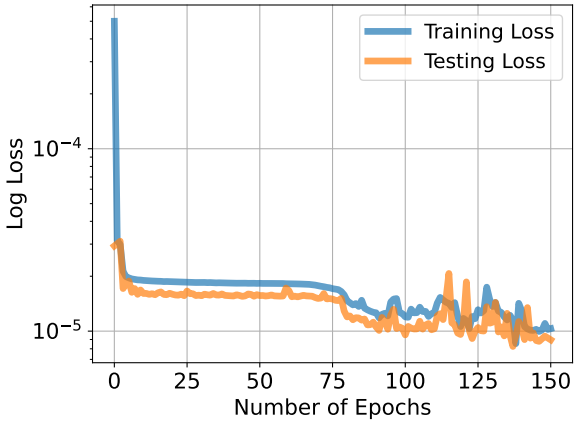


Figure 5.8: Log Loss of Linear_F3_L3

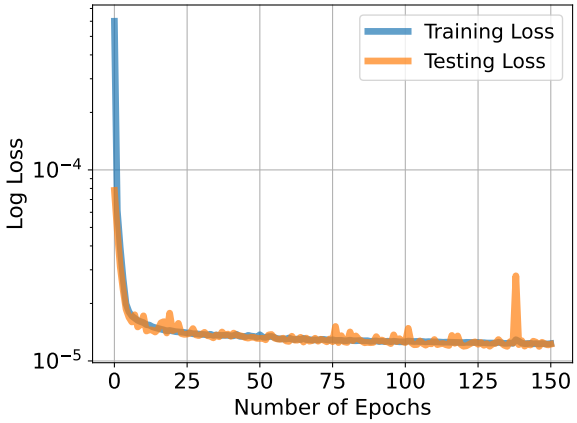


Figure 5.9: Log Loss of LSTM_F3_L3

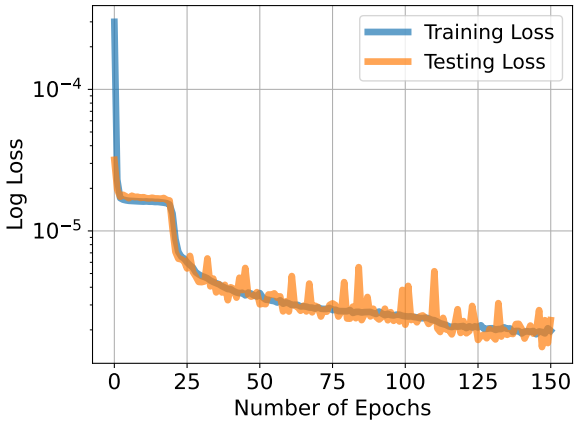


Figure 5.10: Log Loss of Linear_F3_L4

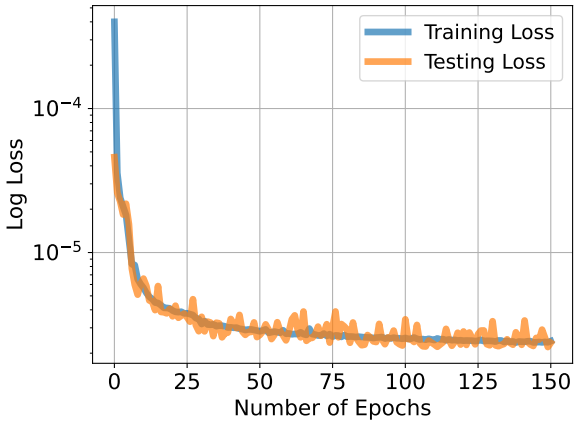


Figure 5.11: Log Loss of LSTM_F3_L4

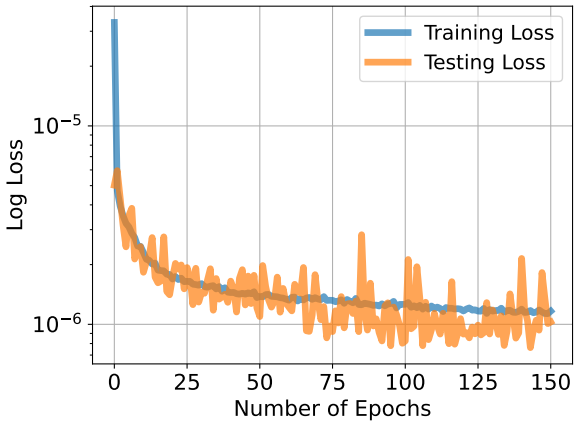


Figure 5.12: Log Loss of Linear_F3_L64

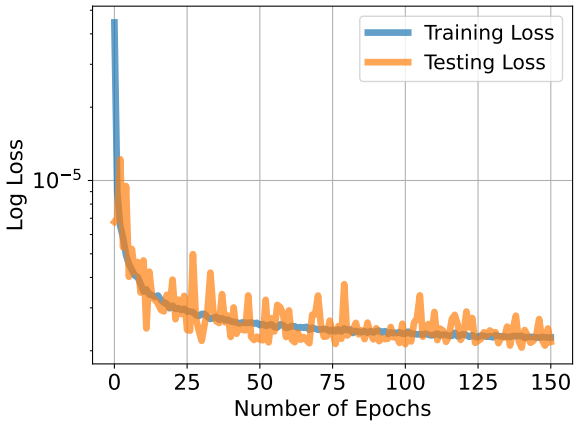


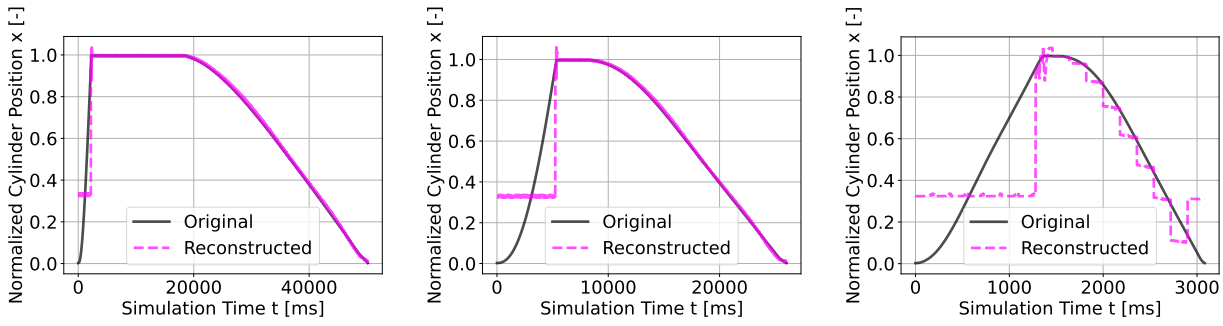
Figure 5.13: Log Loss of LSTM_F3_L64

Table 5.2 lists the loss at the beginning and at the end of each phase of each model. The ”Loss at the beginning” and ”Loss at the end” values are rounded to one decimal, and the ”Reduction percentage” values are rounded to integer.

5.1.2 Reconstruction Result

The 12 Autoencoder models are trained to reduce the dimension of windows with fixed length, and the decoder of the models are trained to reconstruct the windows by the reduced information. For the purpose of visualization and further analysis, the reconstruction figures shown in this section are the reconstructed data of one single data sequence instead of one window. To achieve this, the input data sequence is first cut into windows, and for each window the model generates a reconstructed window, and finally all reconstructed windows are put together to be the reconstructed data sequence.

All data sequences are passed to all AE models. For each model, the reconstruction loss of each input sequence is stored. The mean of these losses are calculated and plotted in fig. 5.14. The recorded reconstruction losses are sorted. The minimum, median, and maximum loss values of each AE model are identified, and the sequences that lead to the minimum, median, and maximum losses are selected and depicted from fig.5.15 to fig.5.44.



(a) (b) (c)

Figure 5.16: Reconstruction of displacement via model Linear_F2_L2. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.

Model Name	Phase	Loss at the beginning	Loss at the end	Reduction percentage (%)
Linear_F2_L2	Training	3.1e−3	3e−4	90
Linear_F2_L2	Testing	3.5e−4	3.3e−4	7
Linear_F2_L4	Training	7.8e−5	3.1e−6	96
Linear_F2_L4	Testing	2.5e−5	3.8e−6	85
Linear_F2_L64	Training	3.9e−5	1.5e−6	96
Linear_F2_L64	Testing	5.9e−6	2.1e−6	65
Linear_F3_L3	Training	5.0e−4	1.0e−5	98
Linear_F3_L3	Testing	2.9e−5	9.0e−6	69
Linear_F3_L4	Training	3.0e−4	2.0e−6	99
Linear_F3_L4	Testing	3.2e−5	2.3e−6	93
Linear_F3_L64	Training	3.3e−5	1.2e−6	96
Linear_F3_L64	Testing	5.0e−6	1.0e−6	80
LSTM_F2_L2	Training	1.5e−3	5.0e−6	100
LSTM_F2_L2	Testing	5.6e−5	5.2e−6	91
LSTM_F2_L4	Training	3.0e−4	5.3e−6	98
LSTM_F2_L4	Testing	4.9e−5	5.4e−6	89
LSTM_F2_L64	Training	4.5e−5	3.4e−6	92
LSTM_F2_L64	Testing	1.4e−5	3.5e−6	75
LSTM_F3_L3	Training	6.0e−4	1.2e−5	98
LSTM_F3_L3	Testing	7.8e−5	1.2e−5	84
LSTM_F3_L4	Training	4.0e−4	2.4e−6	99
LSTM_F3_L4	Testing	4.6e−5	2.4e−6	95
LSTM_F3_L64	Training	4.5e−5	2.3e−6	95
LSTM_F3_L64	Testing	6.8e−6	2.2e−6	68

Table 5.2: Table of loss value in the beginning and the end of the training and testing phase of building the model. The reduction percentage is calculated by $100 * (Loss\ at\ the\ end - Loss\ at\ the\ beginning)/Loss\ at\ the\ beginning$.

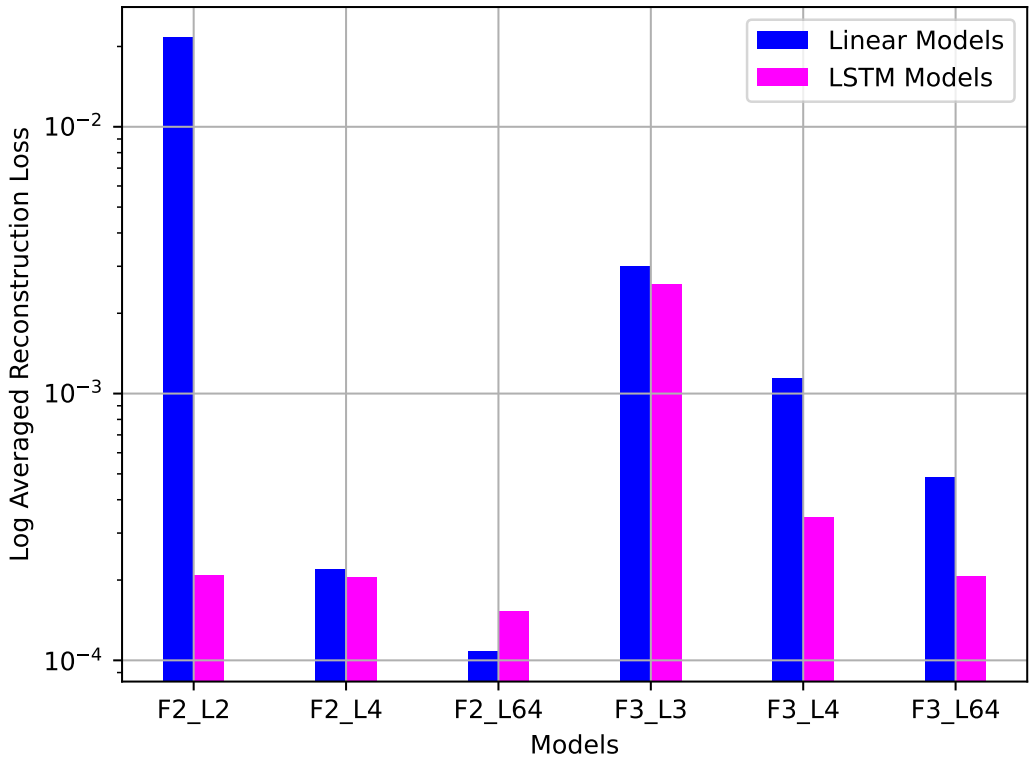


Figure 5.14: Log averaged reconstruction loss of all sequences fed to the AE models.

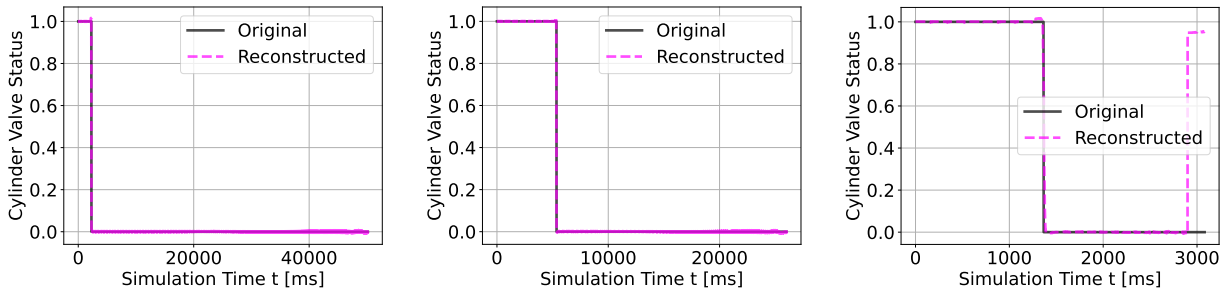
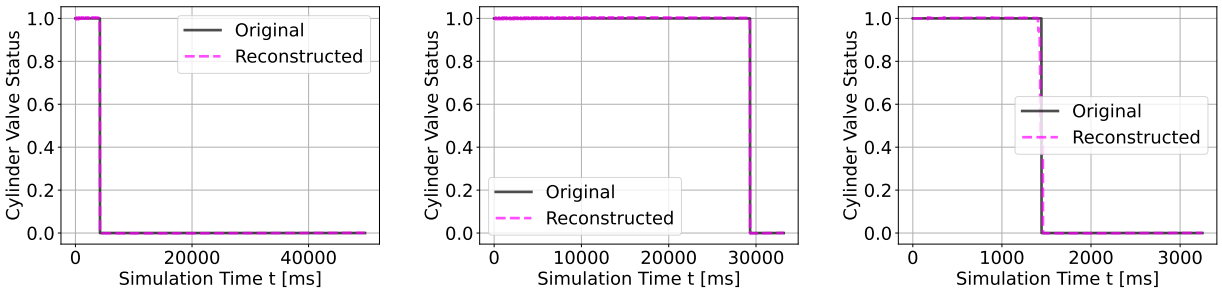
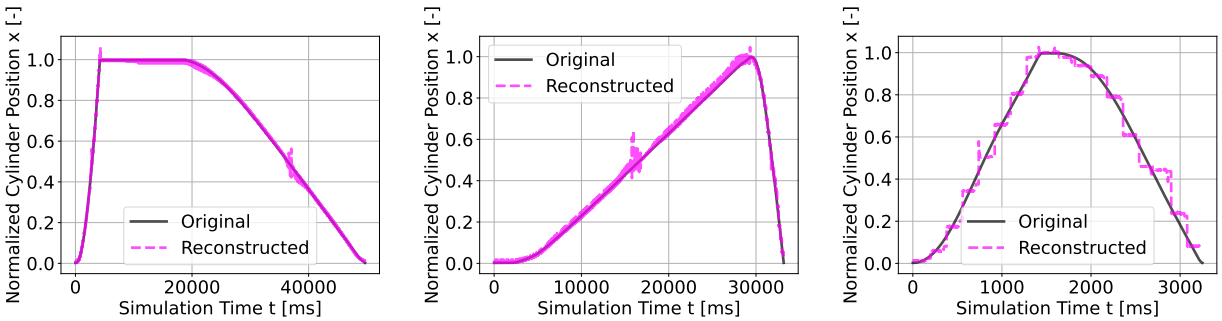


Figure 5.15: Reconstruction of valve status via model Linear_F2_L2. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.



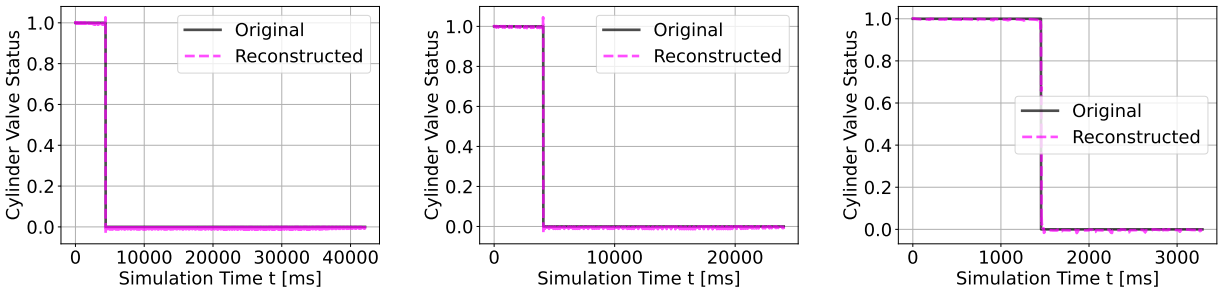
(a) (b) (c)

Figure 5.17: Reconstruction of valve status via model Linear_F2_L4. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.



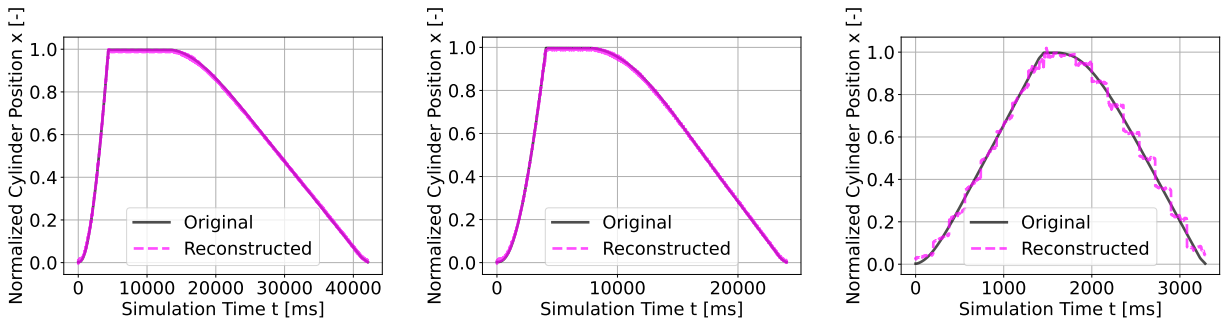
(a) (b) (c)

Figure 5.18: Reconstruction of displacement via model Linear_F2_L4. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.



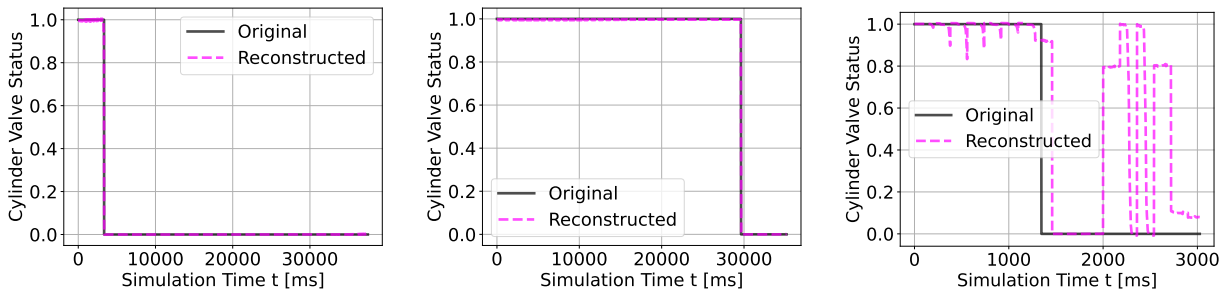
(a) (b) (c)

Figure 5.19: Reconstruction of valve status via model Linear_F2_L64. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.



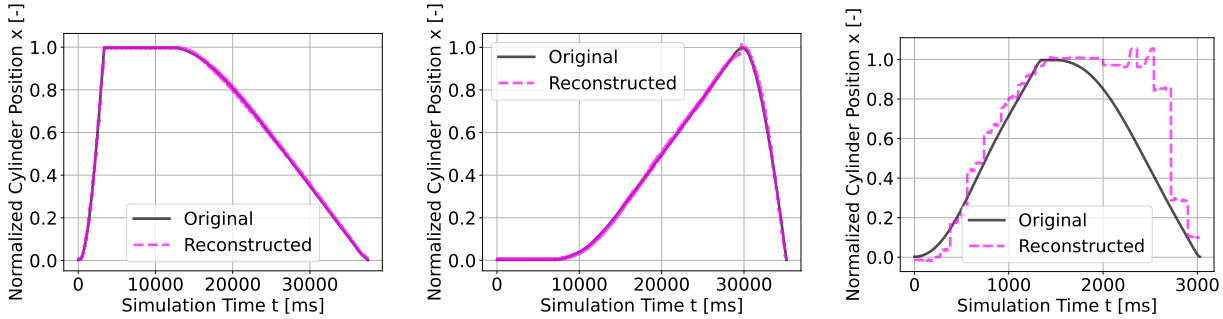
(a) (b) (c)

Figure 5.20: Reconstruction of displacement via model Linear_F2_L64. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.



(a) (b) (c)

Figure 5.21: Reconstruction of valve status via model Linear_F3_L3. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.



(a) (b) (c)

Figure 5.22: Reconstruction of displacement via model Linear_F3_L3. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.

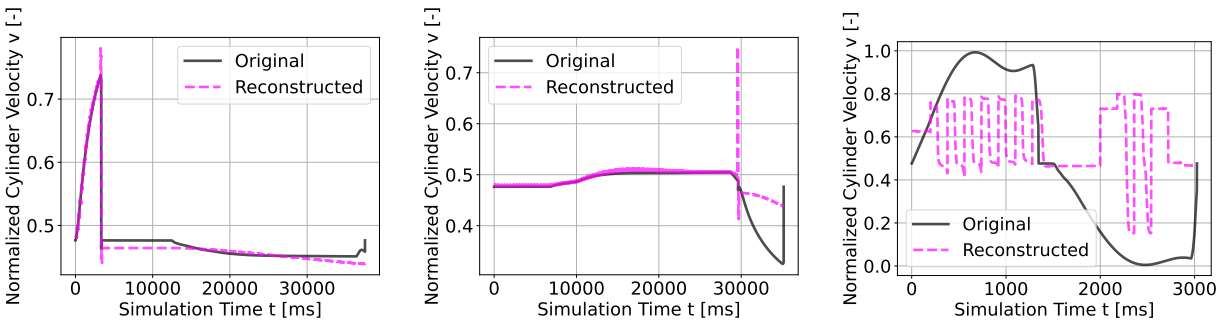


Figure 5.23: Reconstruction of velocity via model Linear_F3_L3. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.

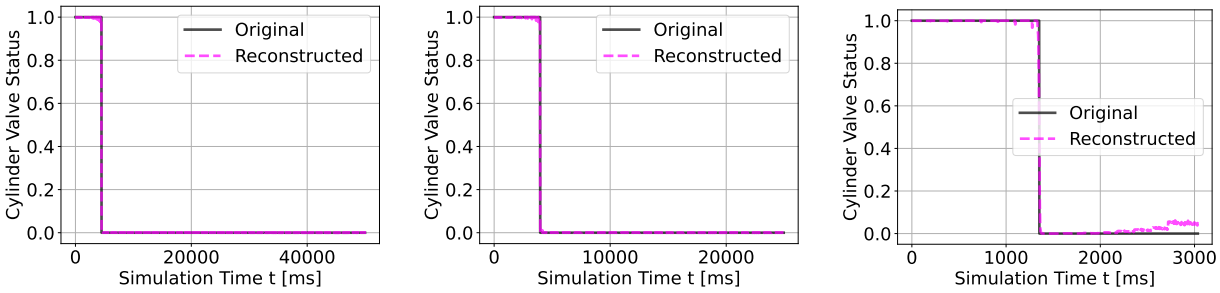


Figure 5.24: Reconstruction of valve status via model Linear_F3_L4. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.

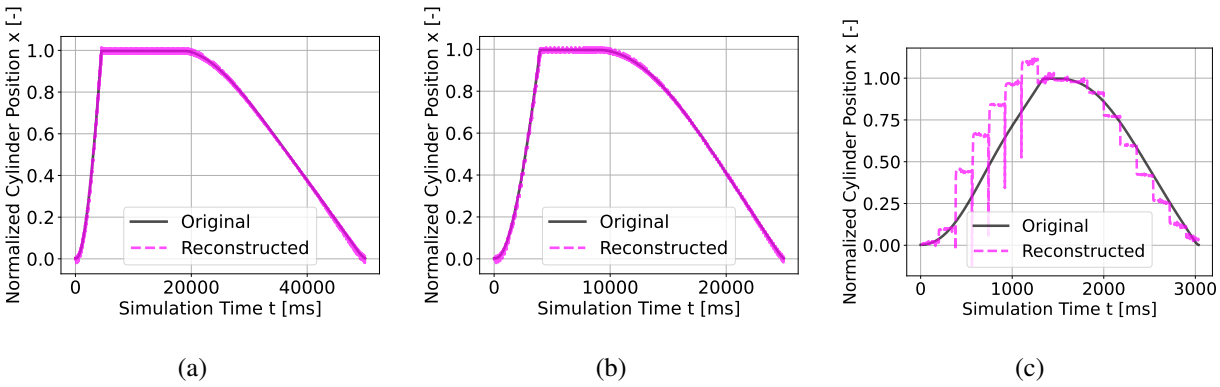
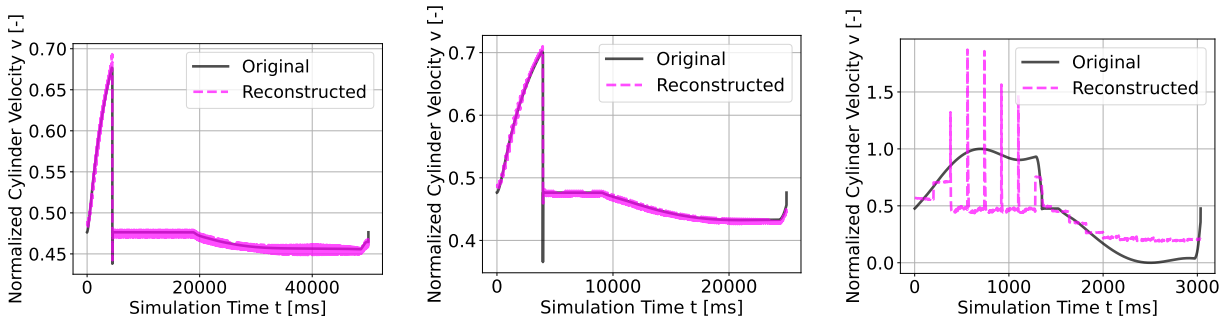
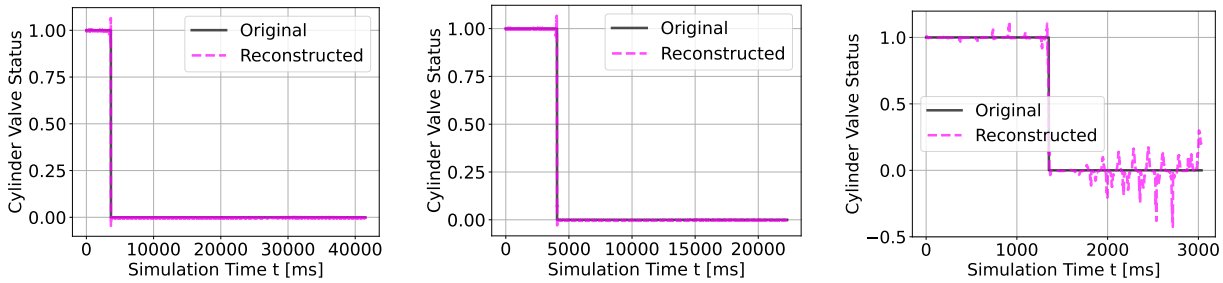


Figure 5.25: Reconstruction of displacement via model Linear_F3_L4. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.



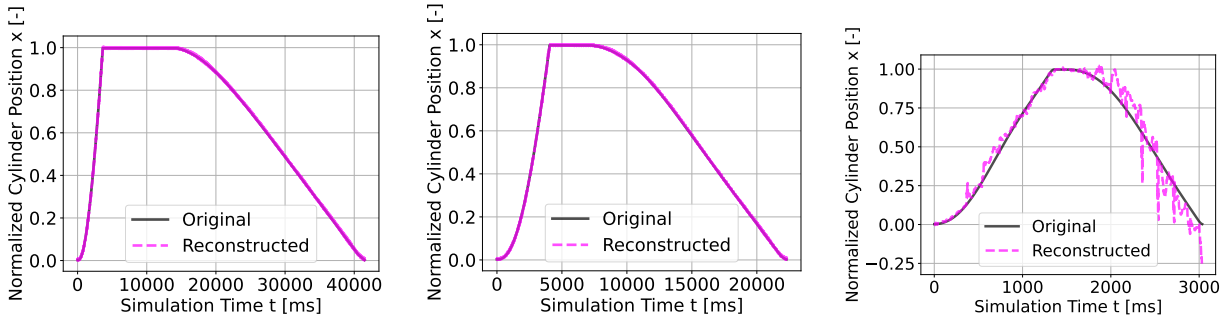
(a) (b) (c)

Figure 5.26: Reconstruction of velocity via model Linear_F3_L4. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.



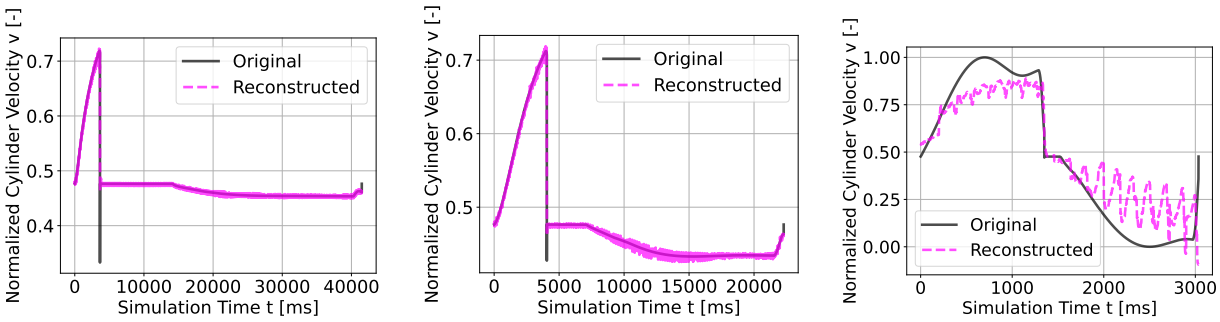
(a) (b) (c)

Figure 5.27: Reconstruction of valve status via model Linear_F3_L64. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.



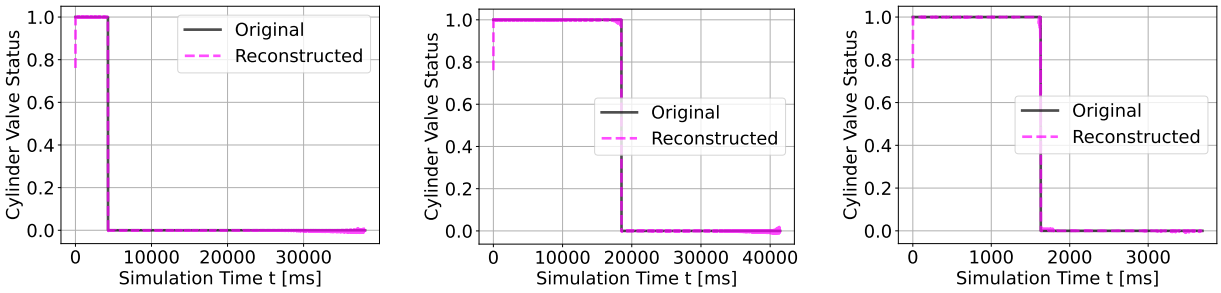
(a) (b) (c)

Figure 5.28: Reconstruction of displacement via model Linear_F3_L64. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.



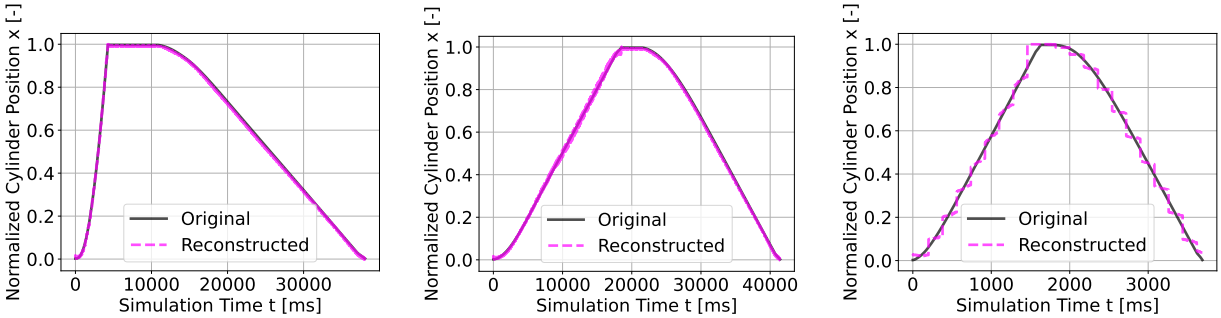
(a) (b) (c)

Figure 5.29: Reconstruction of velocity via model Linear_F3_L64. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.



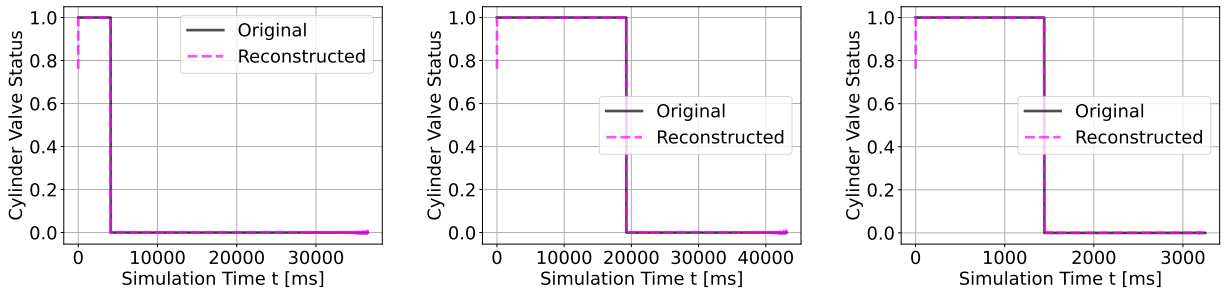
(a) (b) (c)

Figure 5.30: Reconstruction of valve status via model LSTM_F2_L2. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.



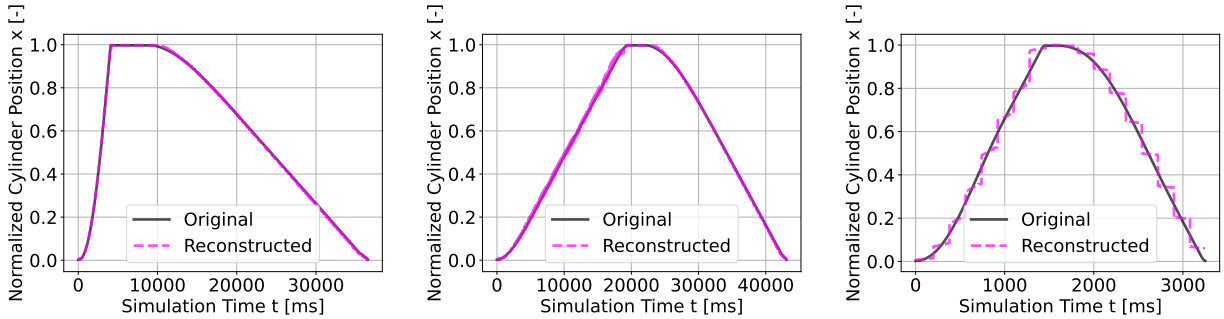
(a) (b) (c)

Figure 5.31: Reconstruction of displacement via model LSTM_F2_L2. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.



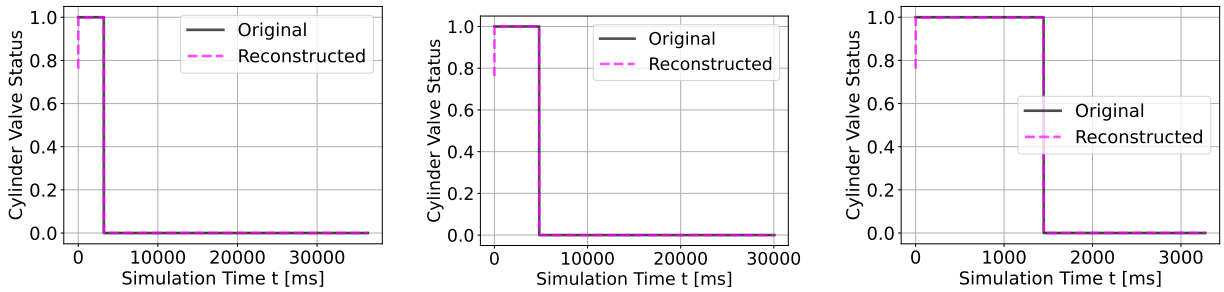
(a) (b) (c)

Figure 5.32: Reconstruction of valve status via model LSTM_F2_L4. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.



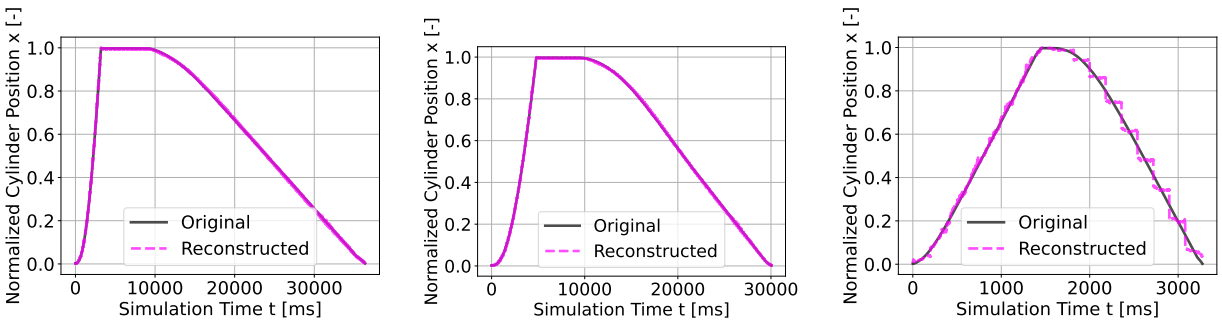
(a) (b) (c)

Figure 5.33: Reconstruction of displacement via model LSTM_F2_L4. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.



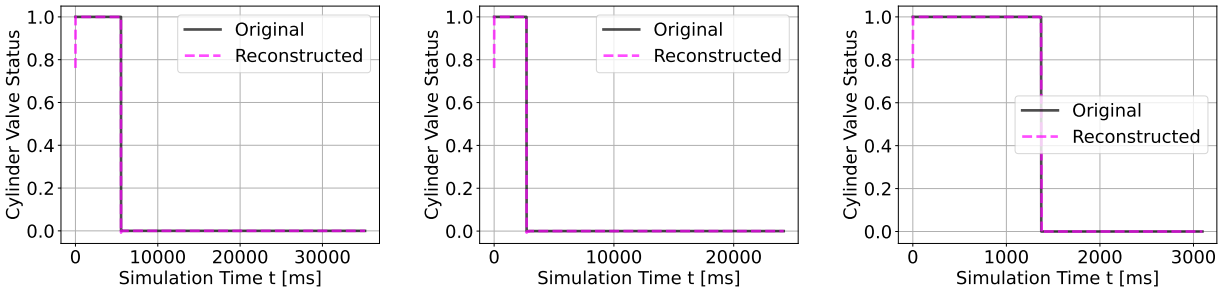
(a) (b) (c)

Figure 5.34: Reconstruction of valve status via model LSTM_F2_L64. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.



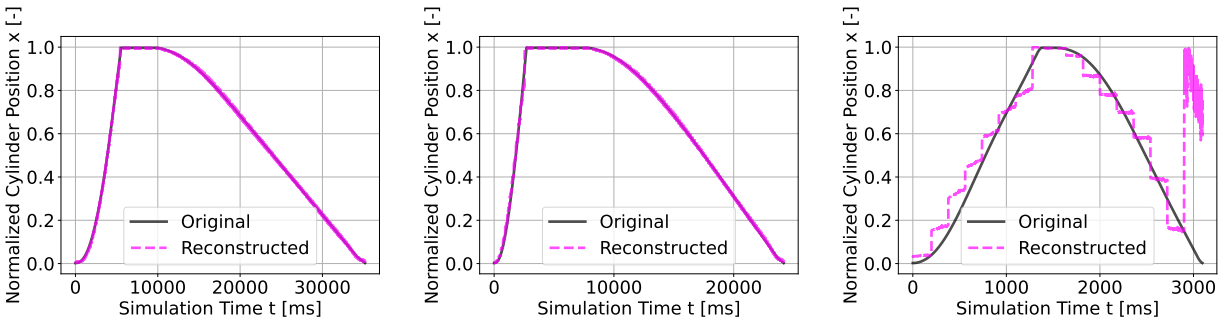
(a) (b) (c)

Figure 5.35: Reconstruction of displacement via model LSTM_F2_L64. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.



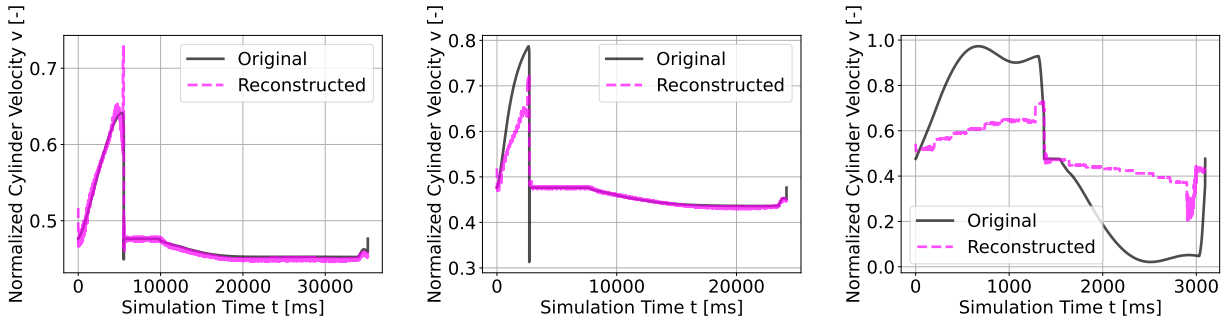
(a) (b) (c)

Figure 5.36: Reconstruction of valve status via model LSTM_F3_L3. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.



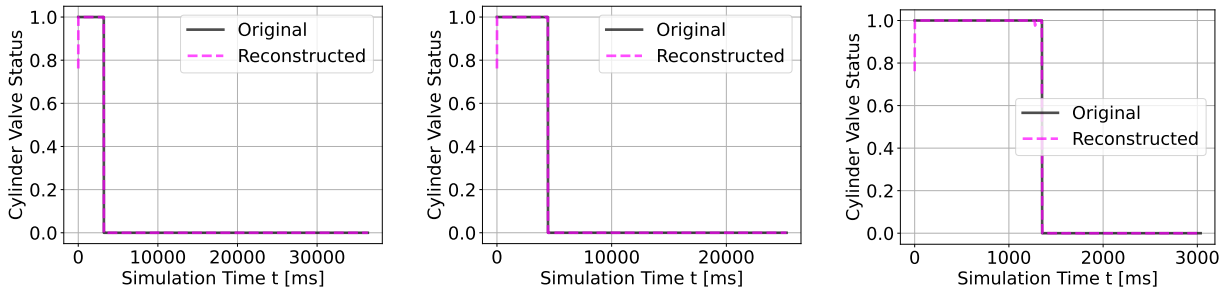
(a) (b) (c)

Figure 5.37: Reconstruction of displacement via model LSTM_F3_L3. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.



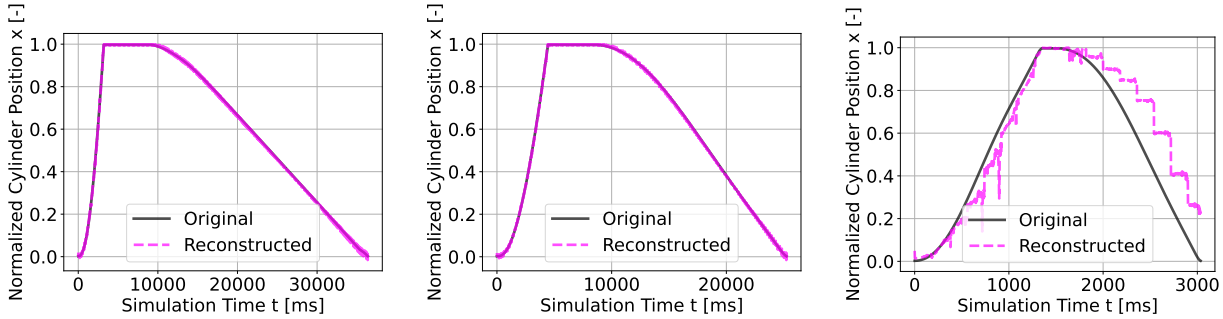
(a) (b) (c)

Figure 5.38: Reconstruction of velocity via model LSTM_F3_L3. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.



(a) (b) (c)

Figure 5.39: Reconstruction of valve status via model LSTM_F3_L4. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.



(a) (b) (c)

Figure 5.40: Reconstruction of displacement via model LSTM_F3_L4. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.

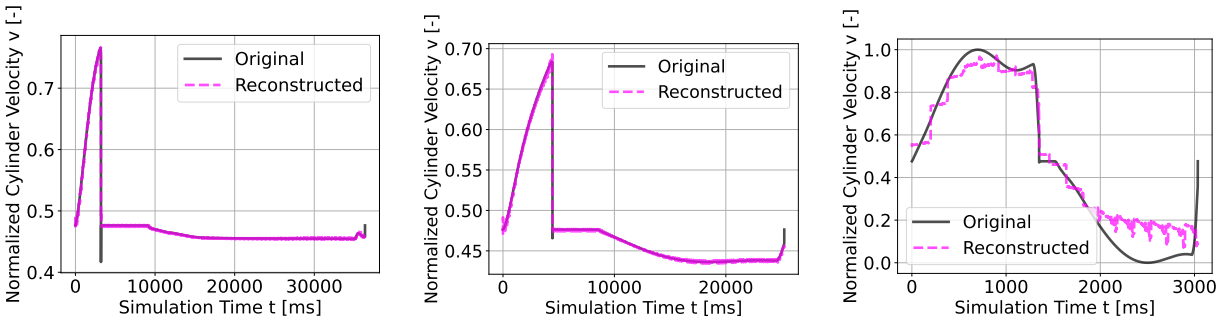


Figure 5.41: Reconstruction of velocity via model LSTM_F3_L4. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.

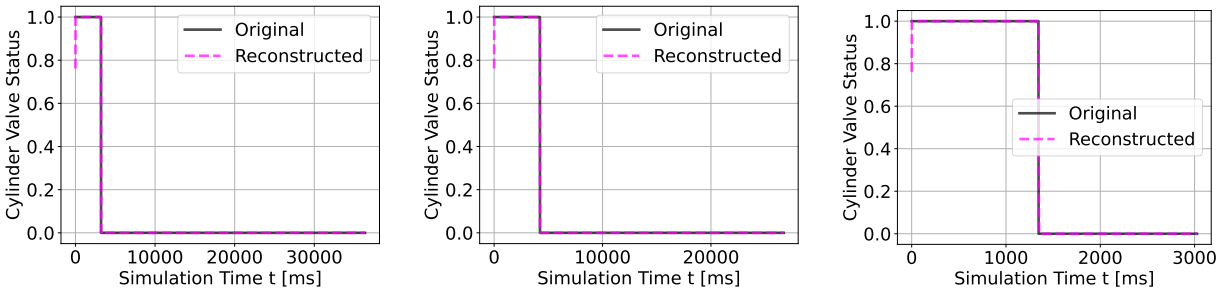


Figure 5.42: Reconstruction of valve status via model LSTM_F3_L64. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.

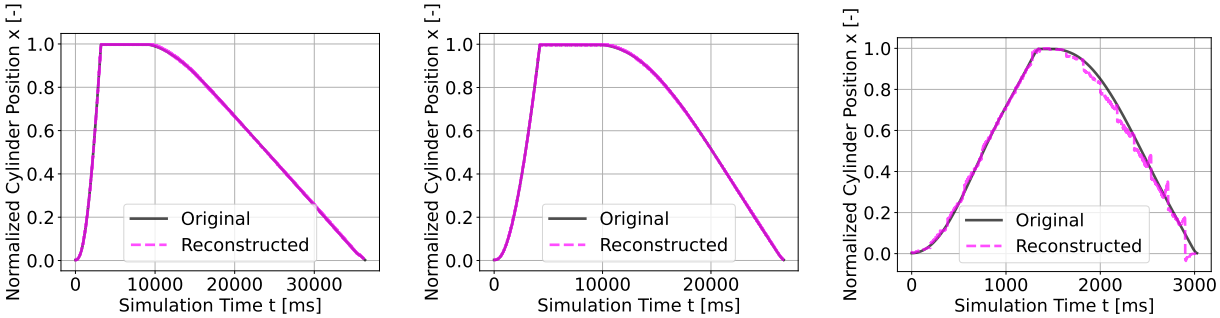


Figure 5.43: Reconstruction of displacement via model LSTM_F3_L64. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.

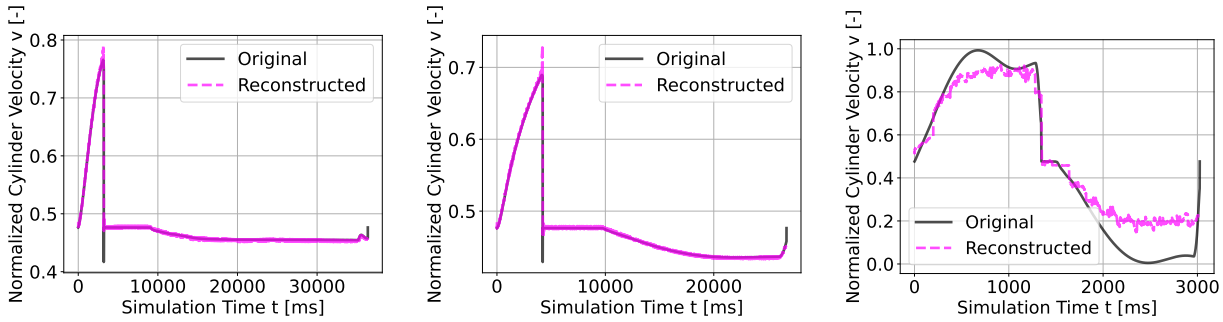


Figure 5.44: Reconstruction of velocity via model LSTM_F3_L64. The data sequences with the (a) minimum, (b) median, and (c) maximum reconstruction loss are selected and depicted.

5.1.3 Classification Result

As explained in sec. 4.3, 48 classifiers are trained with the the lower dimension latent values from 12 autoencoder models. The accuracy of evaluating the 48 classifier models with the testing dataset is presented in table 5.1.3. "Sequence" refers to classifiers trained with data sequences, and "Window" the classifiers trained with data windows. The accuracy values are rounded to integer. Fig. 5.45 depict the accuracy of classifiers trained on latent values from linear autoencoder models, and fig. 5.46 from the LSTM autoencoder models.

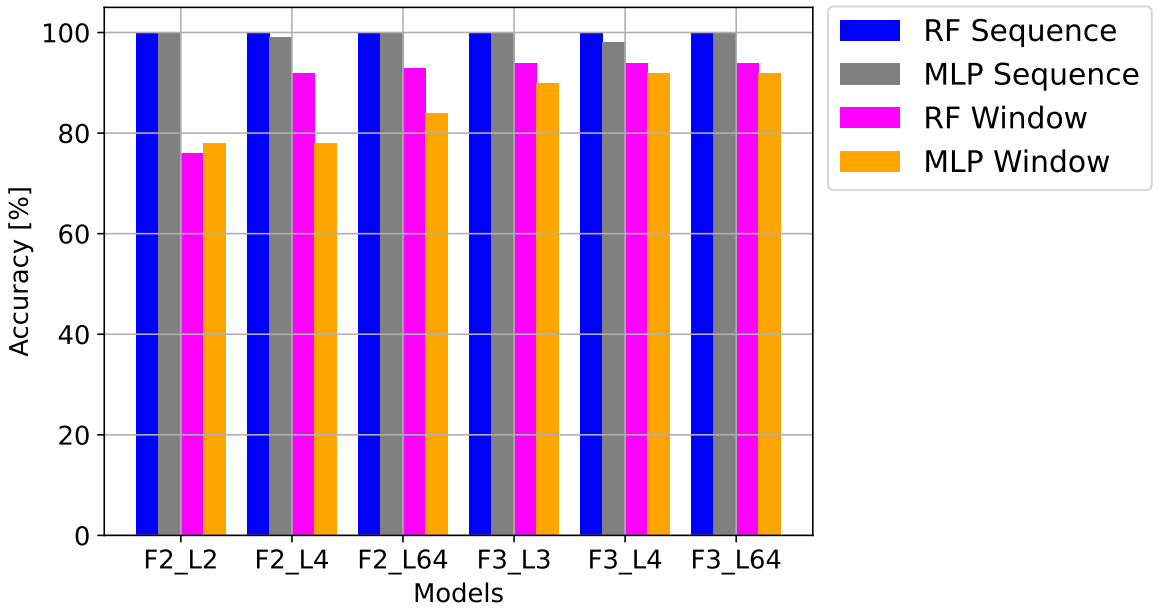


Figure 5.45: Linear Autoencoder Models.

Model Name	RF Sequence [%]	MLP Sequence [%]	RF Window [%]	MLP Window [%]
Linear_F2_L2	100	100	76	78
Linear_F2_L4	100	99	92	78
Linear_F2_L64	100	100	93	84
Linear_F3_L3	100	100	94	90
Linear_F3_L4	100	98	94	92
Linear_F3_L64	100	100	94	92
LSTM_F2_L2	99	98	66	77
LSTM_F2_L4	100	98	91	77
LSTM_F2_L64	100	100	93	91
LSTM_F3_L3	100	100	93	86
LSTM_F3_L4	100	100	94	92
LSTM_F3_L64	100	100	94	92

Table 5.3: Accuracy (%) of classification results using 12 different AE models with two different method of aggregating the latent values.

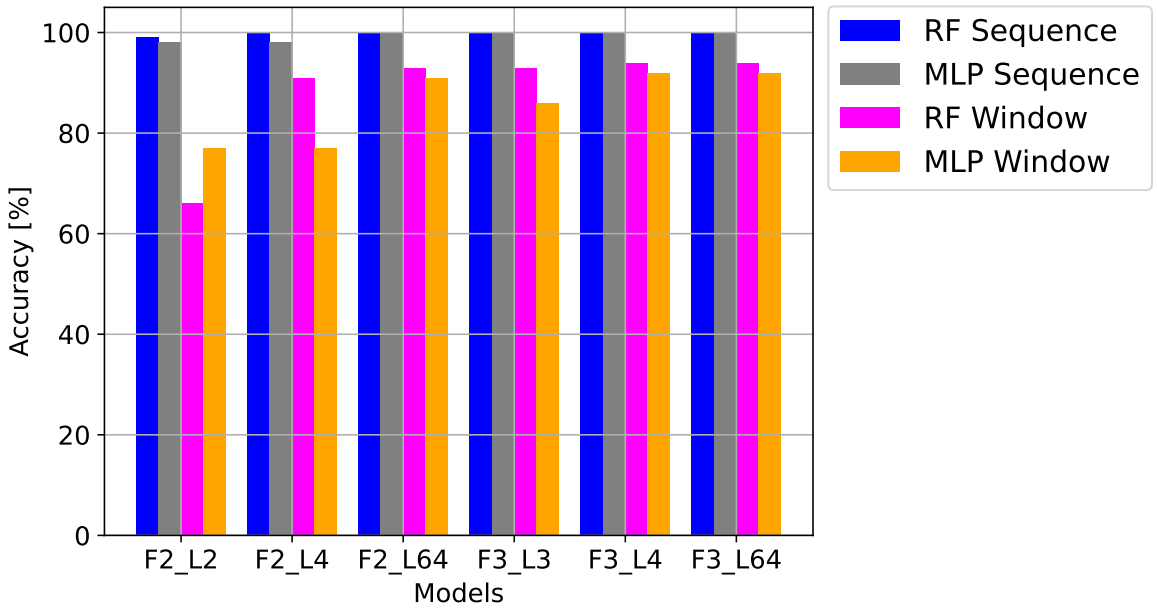


Figure 5.46: LSTM Autoencoder Models.

5.1.4 Latent Values

From fig. 5.47 to fig. 5.52 for the statistics of the latent values. In fig. 5.47 and fig. 5.48, ”Linear. Max.” in the legend means the maximum value in the latent generated from a linear model, and the features size and latent size of such model is described in the horizontal axis. The vertical bar is the range of mean plus and minus one standard deviation. From fig. 5.49 to fig. 5.52, ”W. Max.” in the legend means the maximum value in the latent generated from a each window, and ”S.” stands for averaged latent of a sequence. The features size and latent size of such model is described in the horizontal axis. The vertical bar is the range of mean plus and minus one standard deviation.

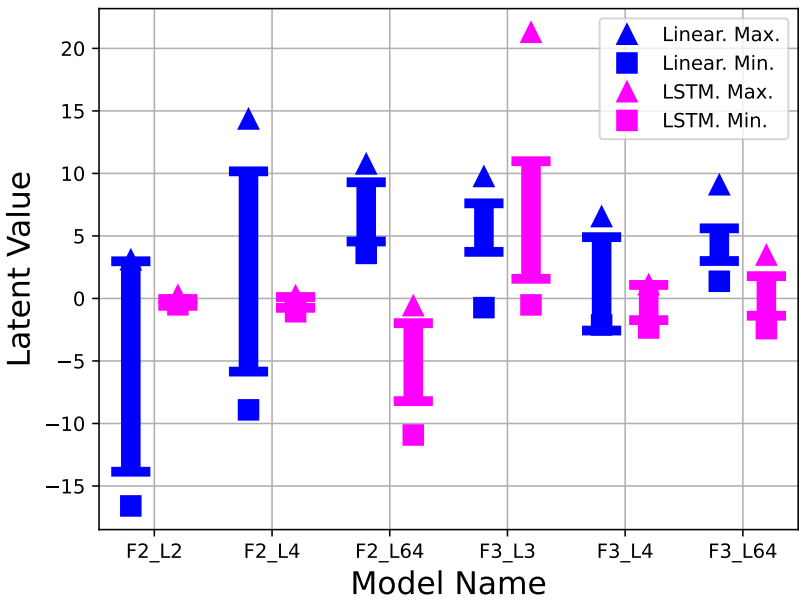


Figure 5.47: Statistics of latent values of each window resulted from all 12 AE models.

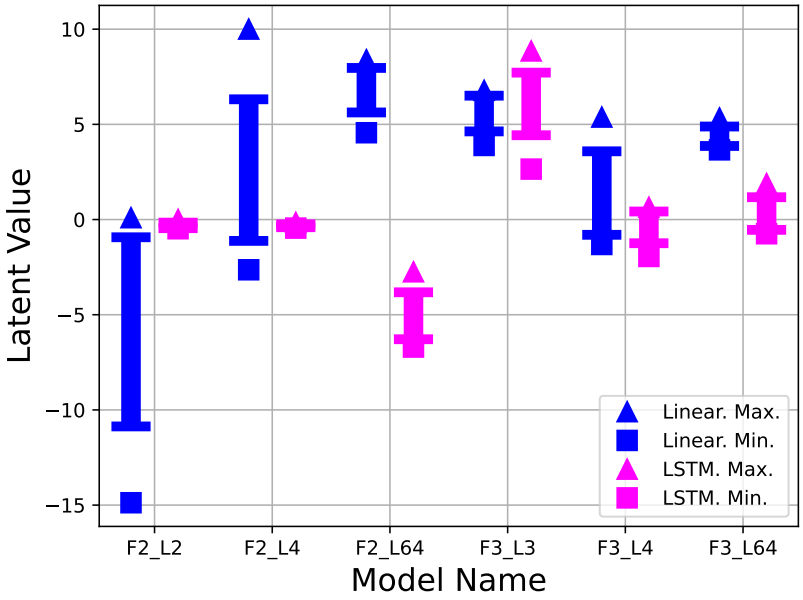


Figure 5.48: Statistics of averaged latent values of each sequence resulted from all 12 AE models.

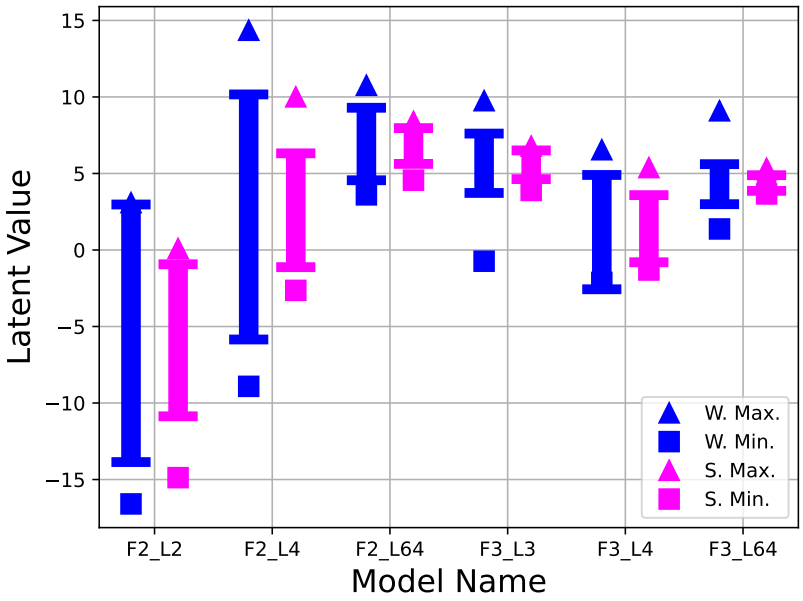


Figure 5.49: Statistics of latent values of resulted from all AE linear models.

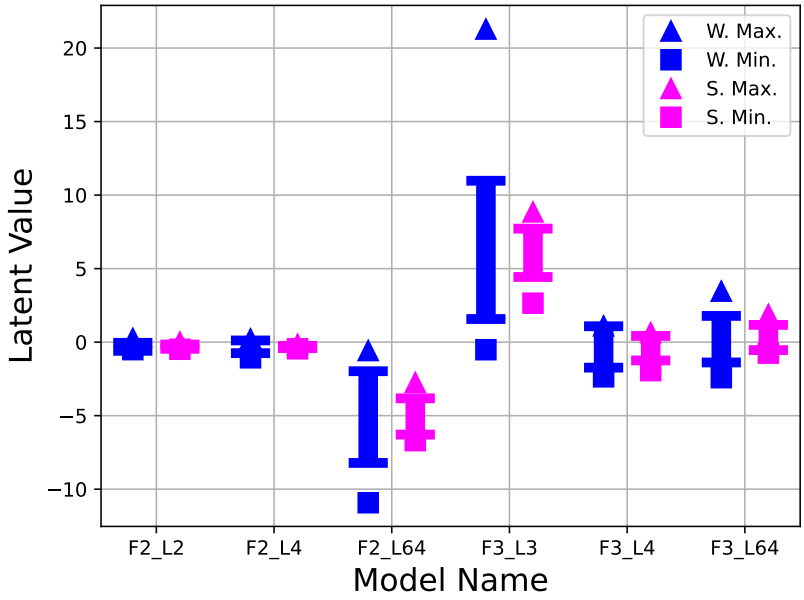


Figure 5.50: Statistics of latent values of resulted from all AE LSTM models.

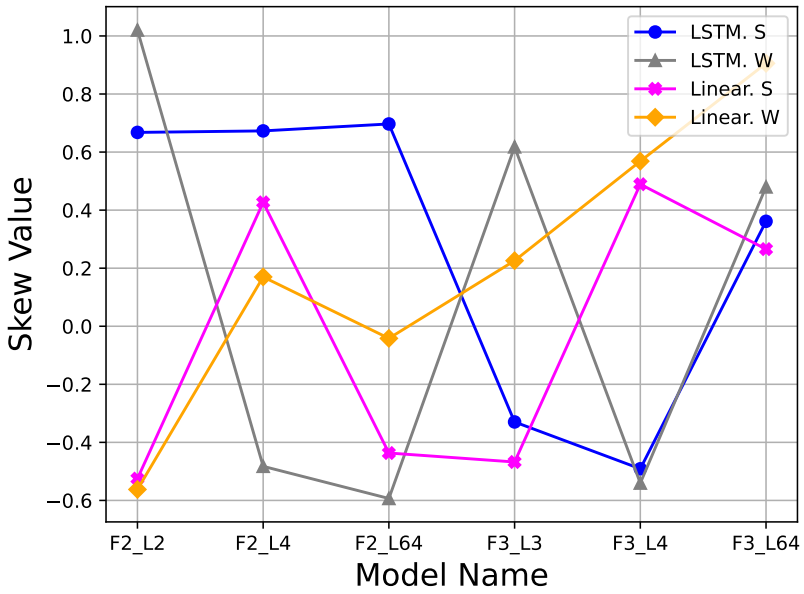


Figure 5.51: Skew of latent values of each window as well as mean latent of each sequence resulted from all 12 AE models.

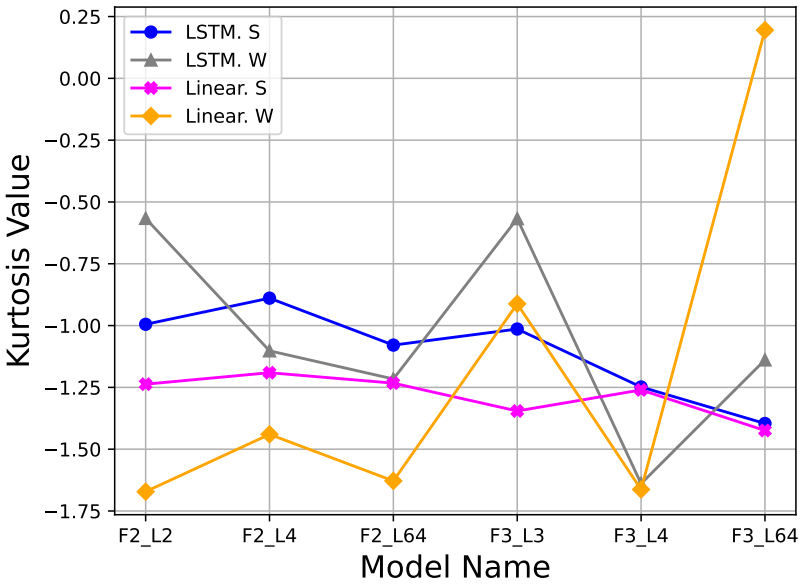


Figure 5.52: Kurtosis of latent values of each window as well as mean latent of each sequence resulted from all 12 AE models.

From fig. 5.53 to fig. 5.64 are the latent values compared between different phases of a sequence. One data sequence is randomly chosen and then manually classify each window into phases: increasing, holding, or decreasing.

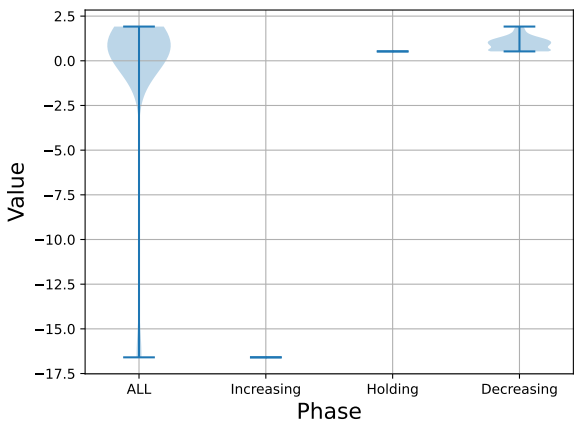


Figure 5.53: Linear_F2_L2

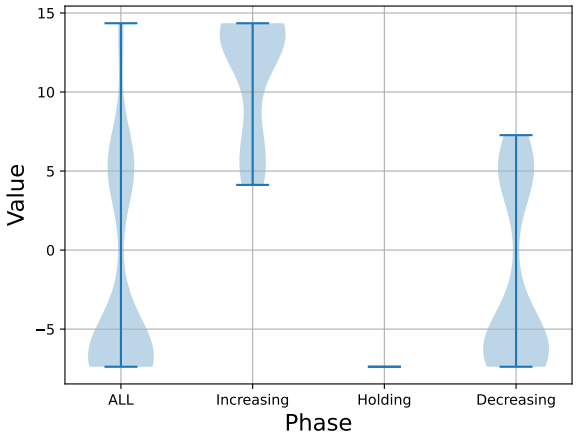


Figure 5.54: Linear_F2_L4

In order to observe the relationship between the latent values of different sequences and the leakage status, averaged latent values of sequences are depicted from fig. 5.65 to fig. 5.72. For latent values with 4 dimensions, 3 of the dimensions that can separate the latent values of different leakage category the most distinctly are selected for plotting.

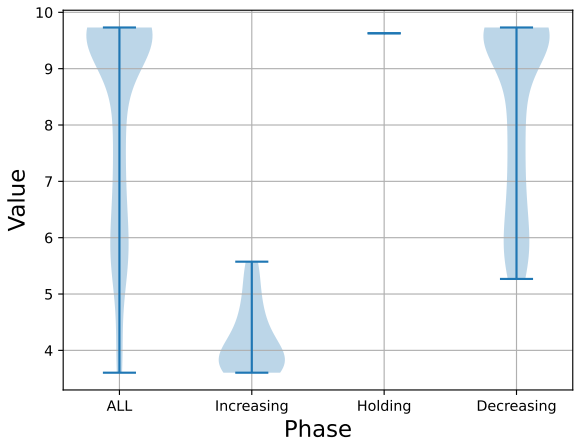


Figure 5.55: Linear_F2_L64

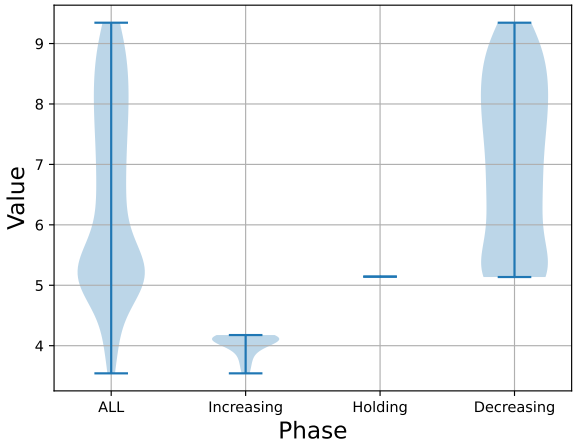


Figure 5.56: Linear_F3_L3

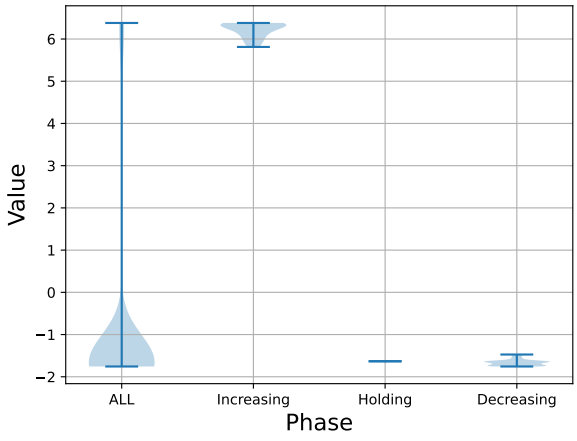


Figure 5.57: Linear_F3_L4

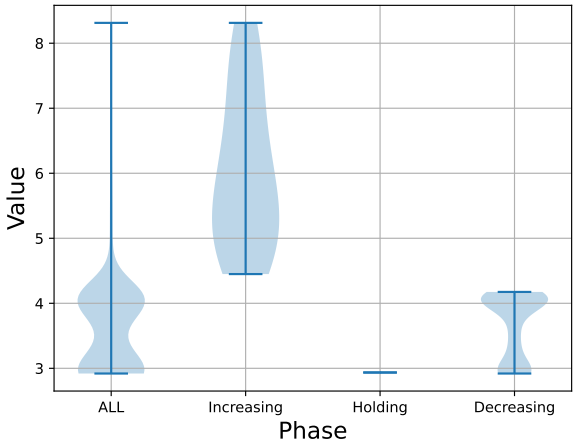


Figure 5.58: Linear_F3_L64

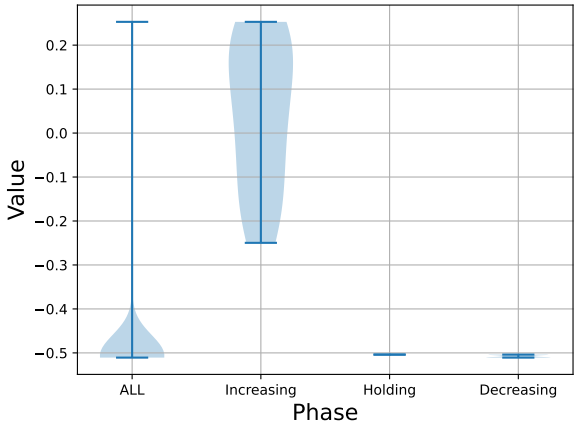


Figure 5.59: LSTM_F2_L2

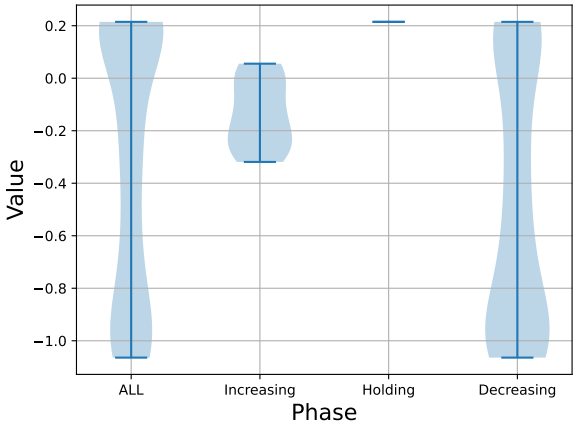


Figure 5.60: LSTM_F2_L4

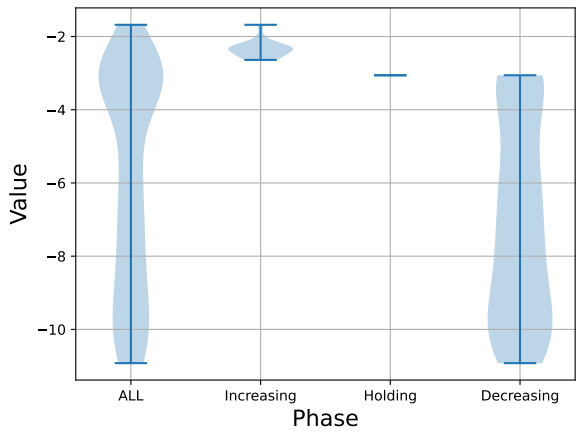


Figure 5.61: LSTM_F2_L64

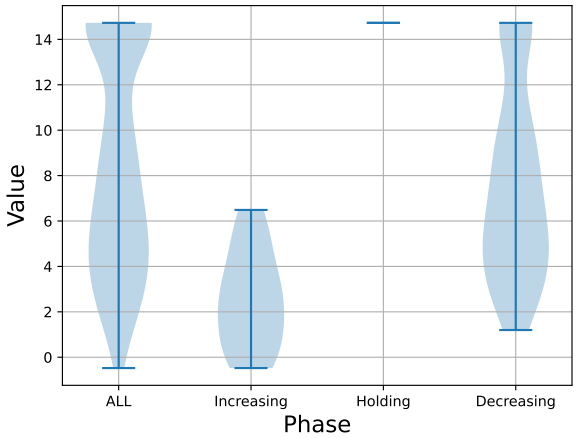


Figure 5.62: LSTM_F3_L3

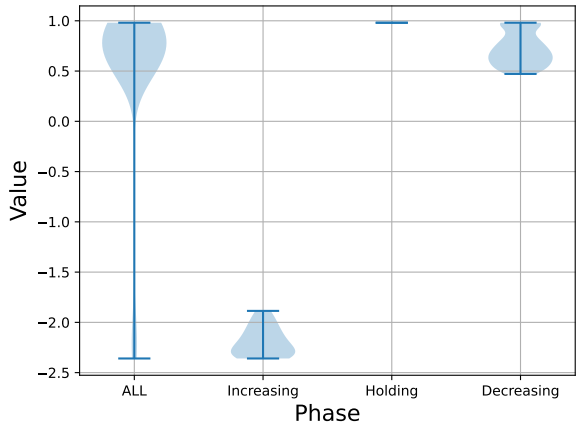


Figure 5.63: LSTM_F3_L4

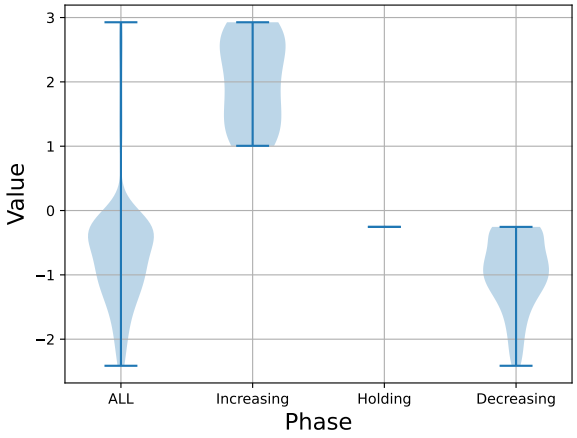


Figure 5.64: LSTM_F3_L64

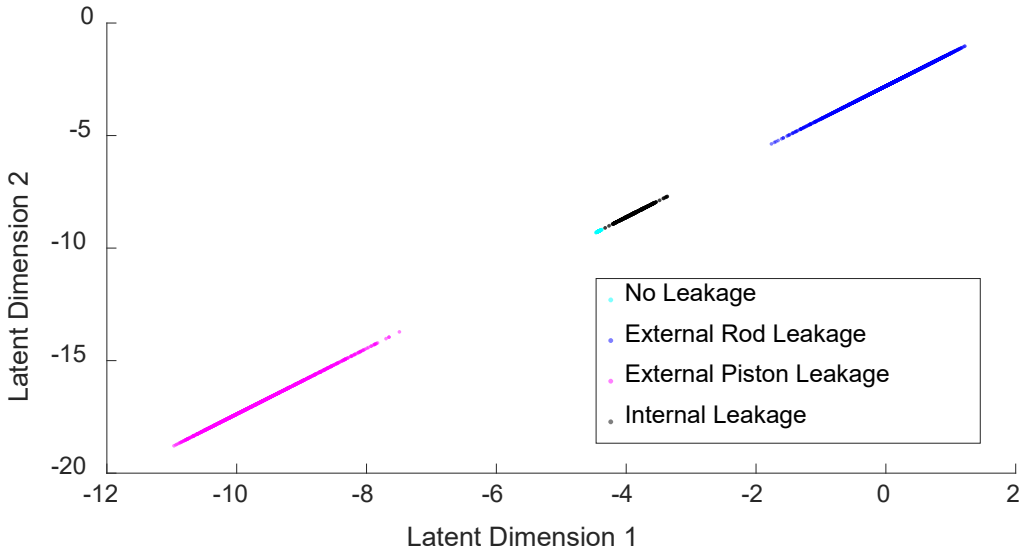


Figure 5.65: The averaged latent values of data sequences which are compressed using AE model Linear_F2_L2.

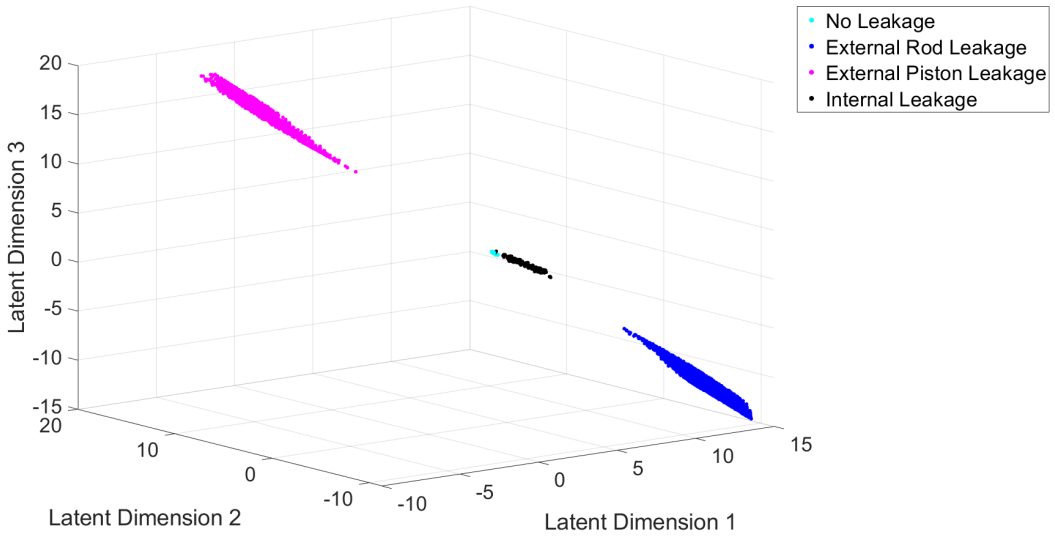


Figure 5.66: The averaged latent values of data sequences which are compressed using AE model Linear_F2_L4.

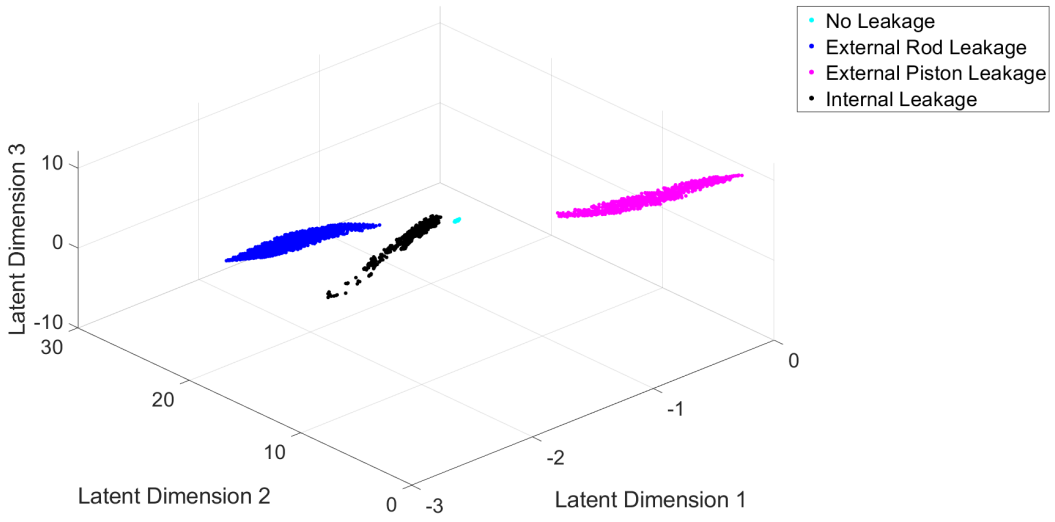


Figure 5.67: The averaged latent values of data sequences which are compressed using AE model Linear_F3_L3.

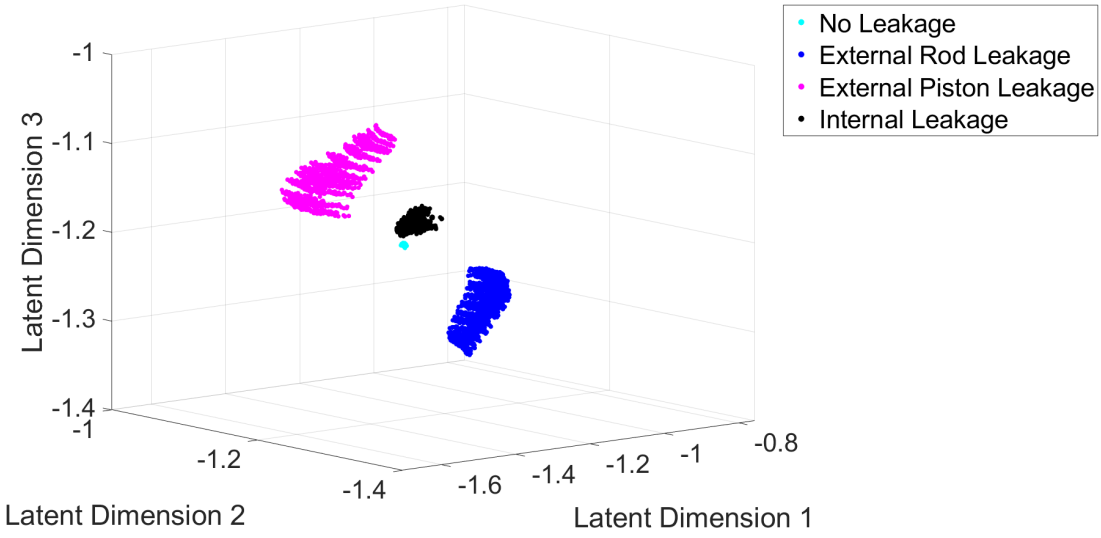


Figure 5.68: The averaged latent values of data sequences which are compressed using AE model Linear_F3_L4.

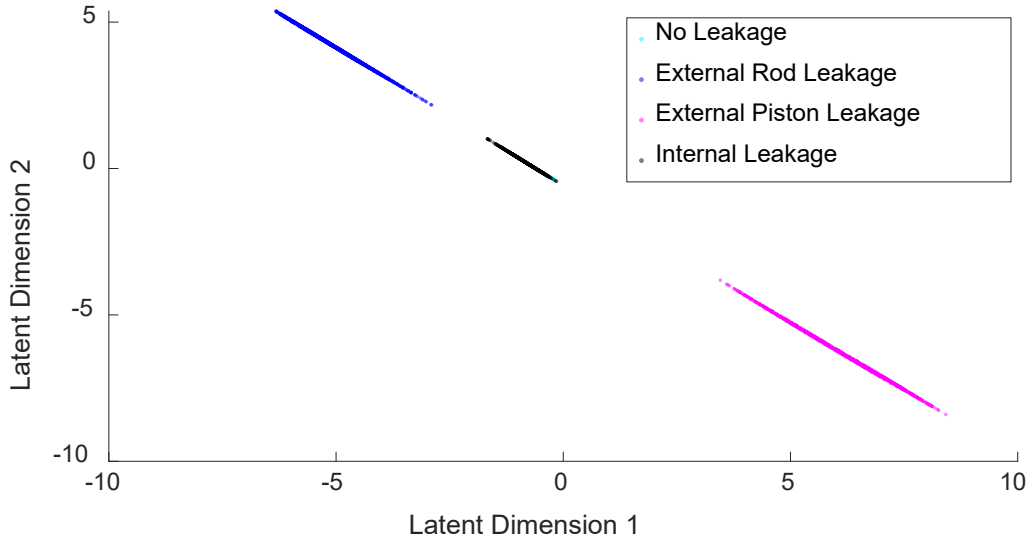


Figure 5.69: The averaged latent values of data sequences which are compressed using AE model LSTM_F2_L2.

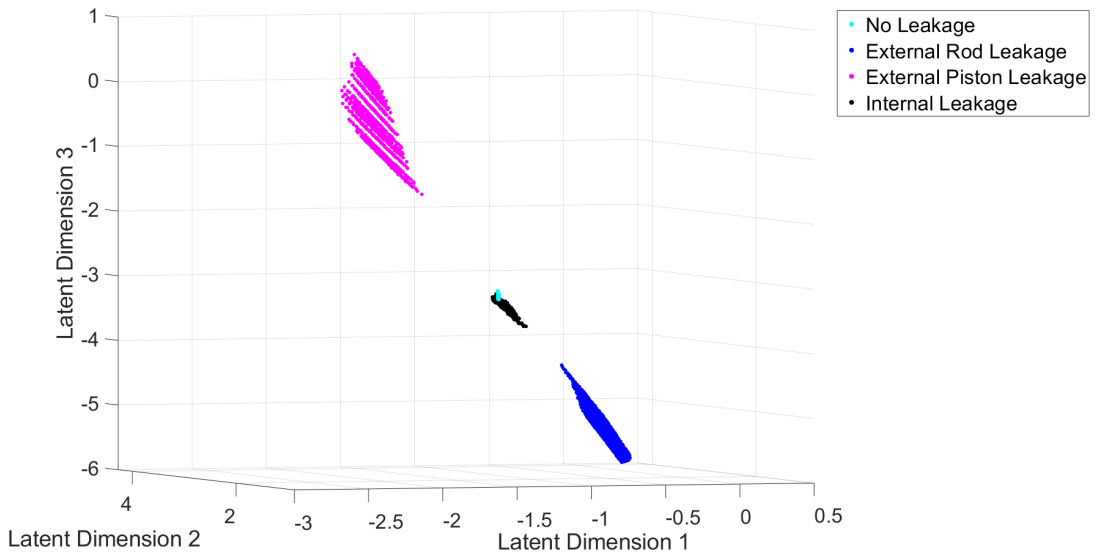


Figure 5.70: The averaged latent values of data sequences which are compressed using AE model LSTM_F2_L4.

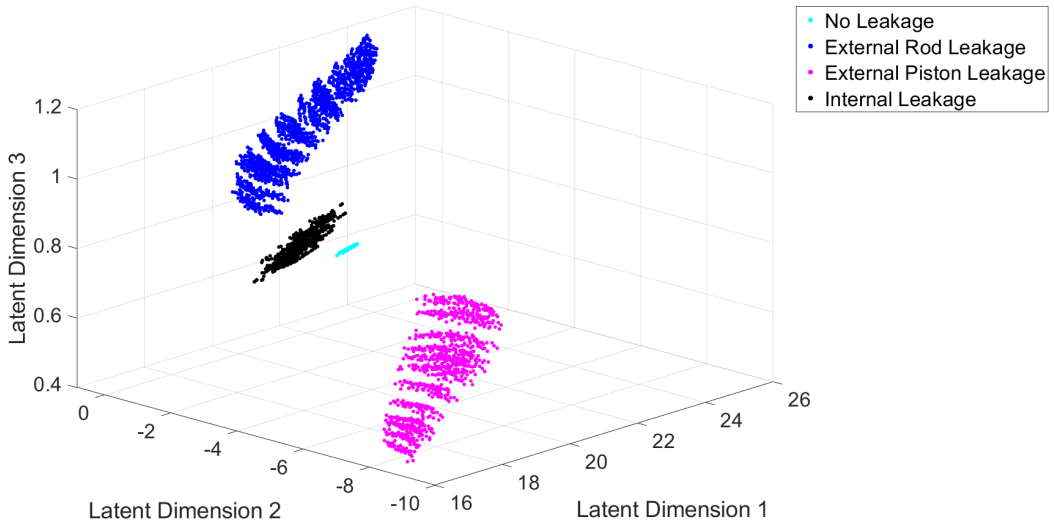


Figure 5.71: The averaged latent values of data sequences which are compressed using AE model LSTM_F3_L3.

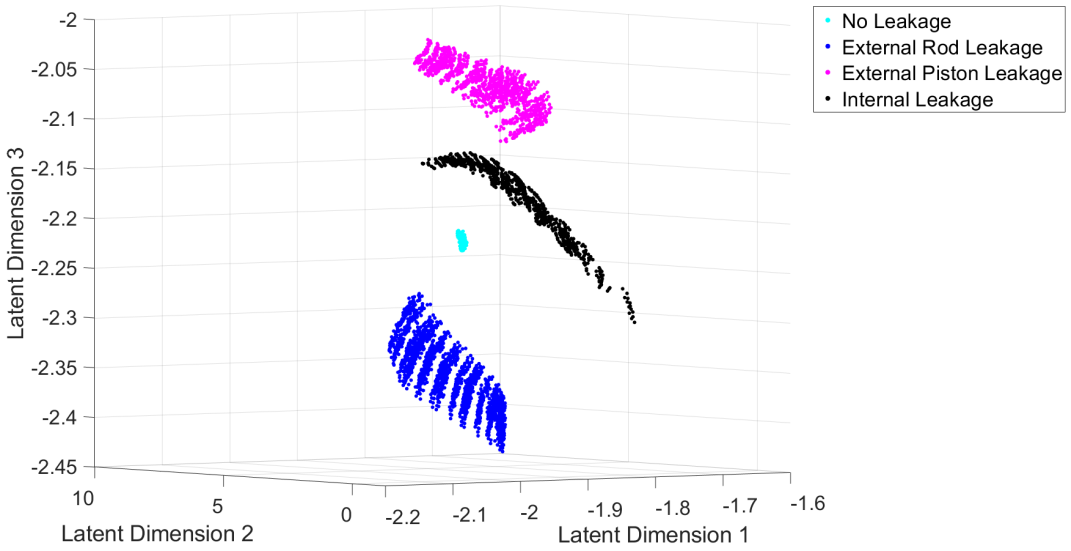


Figure 5.72: The averaged latent values of data sequences which are compressed using AE model LSTM_F3_L4.

5.2 Variational Autoencoder

This research focus mostly on the investigation of AE models, and therefore does not implement as many different VAE models as the number of AE models. Two VAE models are researched: a basic VAE model ans a self-consistent VAE model. The architectures of these models can be found in sec. 4.4. The reconstruction results of the basic VAE model are shown from fig. 5.73 to fig. 5.75, and from fig. 5.76 to fig. 5.78 are the reconstruction plots of the self-consistent VAE model. These models shown are trained with 3 input features (including velocity) and the latent dimension is 3 as well. The reconstructed data are completely different from the original data. VAE models with different input features and latent dimensions are also carried out, but the reconstructed data are still far from the original.

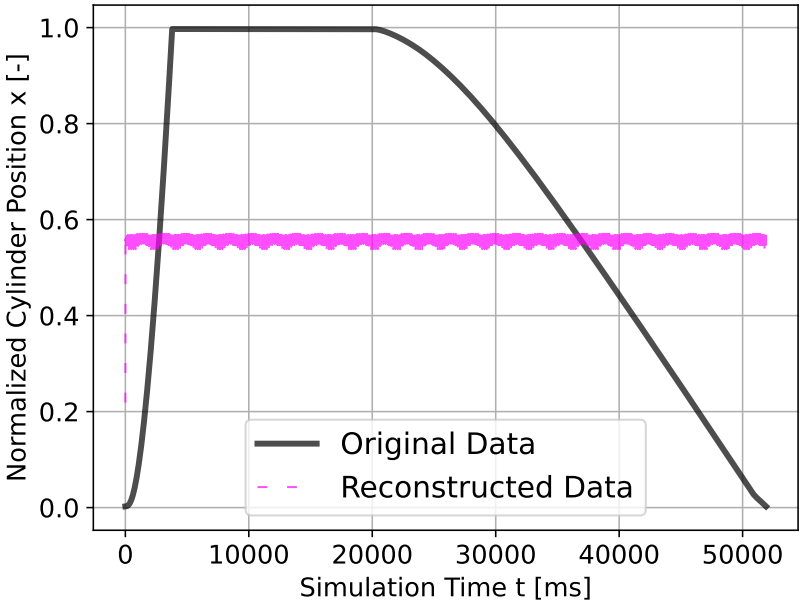


Figure 5.73: Cylinder displacement data reconstructed via basic VAE model compared with original data.

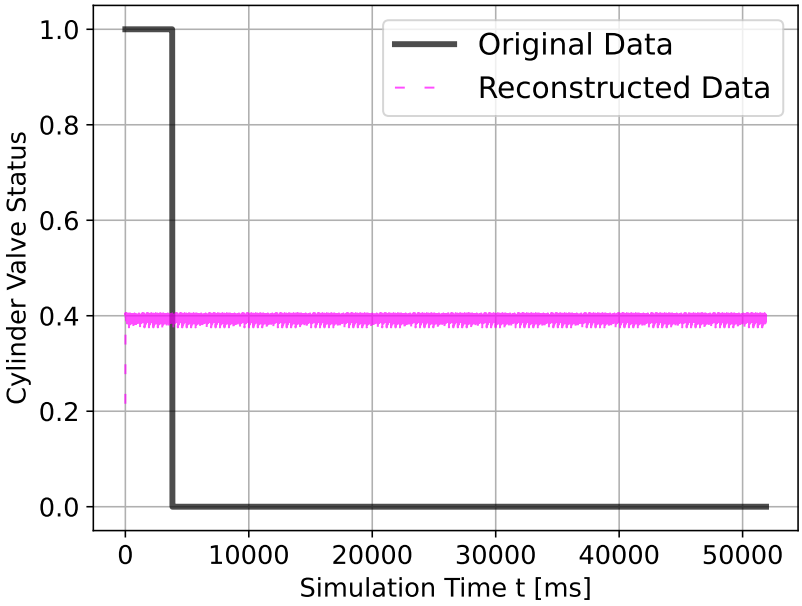


Figure 5.74: Valve status data reconstructed via basic VAE model compared with original data.

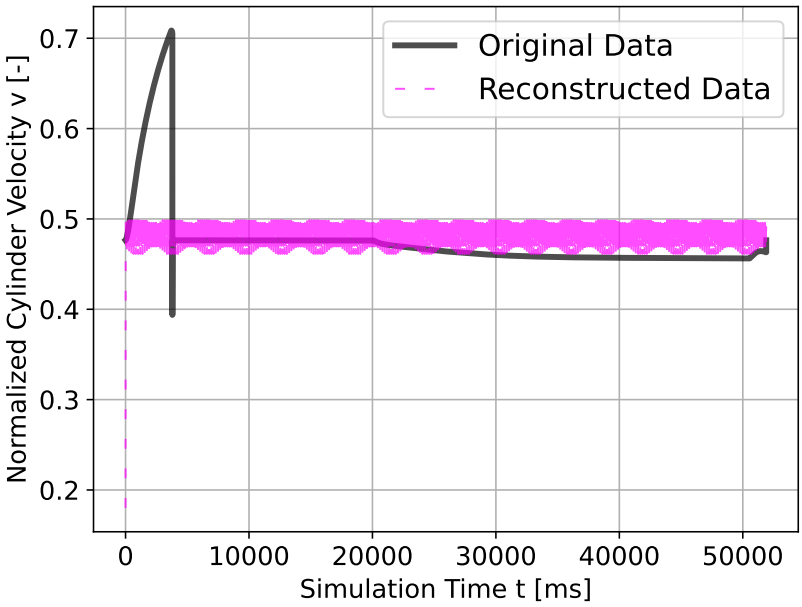


Figure 5.75: Velocity data reconstructed via basic VAE model compared with original data.

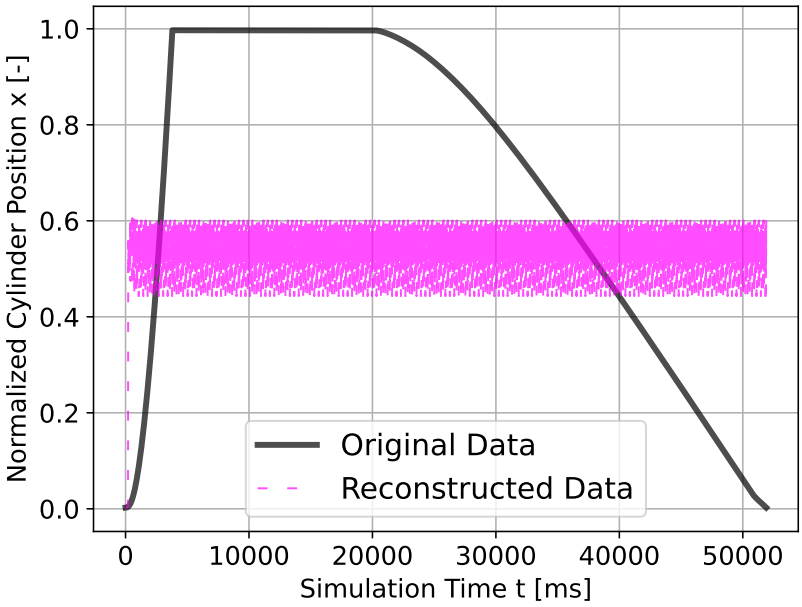


Figure 5.76: Cylinder displacement data reconstructed via self-consistent VAE model compared with original data.

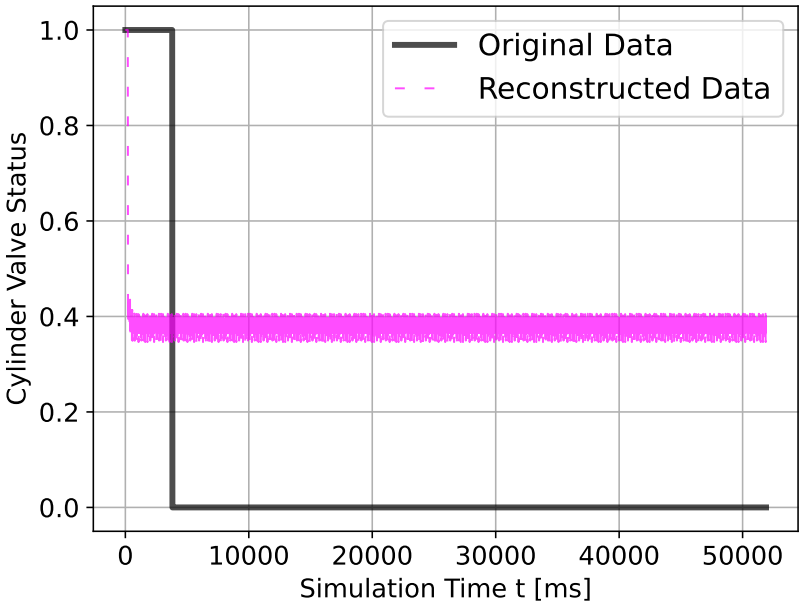


Figure 5.77: Valve status data reconstructed via self-consistent VAE model compared with original data.

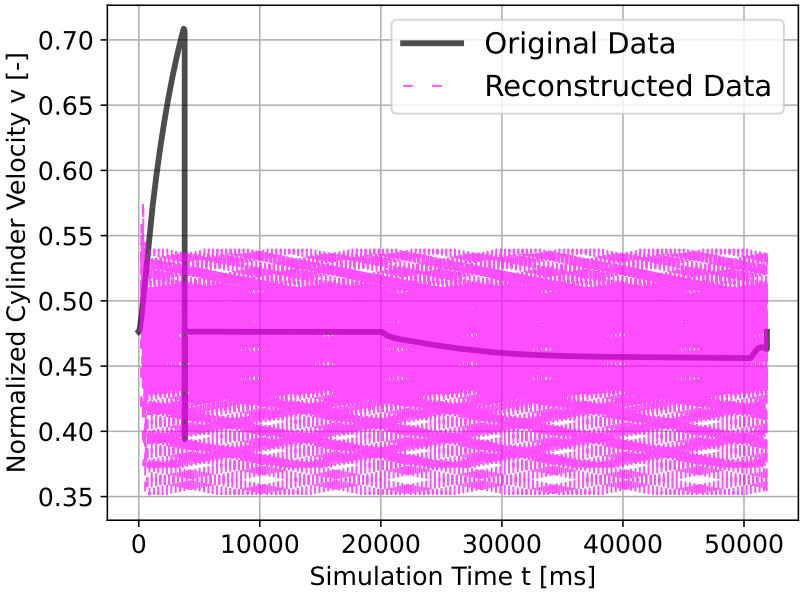


Figure 5.78: Velocity data reconstructed via self-consistent VAE model compared with original data.

6 Discussion

This chapter presents further analysis and discussion on the results shown in chapter 5. Sec. 6.1 is about results regarding the AE models, while sec. 6.2 about the VAE models. Due to the promising results of the AE models, deeper analyses are carried out in sec. 6.1. The meaning of training and testing loss is explained, as well as how to determine whether a model performs well. Reconstructed data is another important indicator of the performance of AE models, and comparisons are made between different models. Classification results are crucial for condition monitoring, the eventual goal of dimension reduction in this thesis. Latent space analysis provides an understanding of the encoded data generated by the UL models, revealing more insight into the models and the model training pipeline.

6.1 Autoencoder

In this section, the results of autoencoder models in sec. 5.1 are analyzed. Loss, reconstructed data, and classification results are three indicators of the models. In the loss analysis section, it can be seen that model Linear_F2_L2 may not be a good choice. When comparing the reconstructed data with the original data, model Linear_F2_L2 and model Linear_F3_L3 fail to reconstruct to original data. For the classification part, model Linear_F2_L2 and Linear_F3_L3 perform the worst. This research therefore assume that latent size should be at least 4 to guarantee a good performance in both reconstructing and classifying. When the latent size is less than 4, it may not be sufficient to accomplish the reconstructing task of the classifying task or both. Furthermore, when latent size is at least 4, obvious advantages of using LSTM models cannot be observed, while when latent size is less than 4, the LSTM models perform better.

6.1.1 Loss Result Analysis

By observing the training loss and the testing loss, overfitting and underfitting problems can be detected. If the training loss keeps decreasing with epochs and doesn't slow down, this means that 151 epochs are not enough for training the model. From the figures in sec. 5.1.1, it can be seen that all training loss plots reach a almost horizontal line, meaning that 151 epochs are enough for training

these models and increasing number of epochs wouldn't change the result much. Overfitting problem is that the model learns the training data too well and will therefore produce very good results on the training data but much worse results on the testing data. The testing curves in sec. 5.1.1 are in the similar level of the training curve, meaning these models don't have the overfitting problem. In addition, it can be seen from table 5.2 that most model can reduce the training loss by more than 90%, meaning that these model are indeed learning from the dataset.

Comparing reduction percentage between the training and testing loss, the reduction percentages of testing loss are lower. The first training loss comes from the first epoch of the training process, so the model has not have much knowledge about the data yet. On the other hand, the first testing loss is calculated by feeding the testing dataset to the first model trained by the training dataset, therefore the initial testing loss is lower than the initial training loss, and the reduction percentage is lower as well.

The reduction percentage of the testing loss of the model Linear_F2_L2 is much lower than the other models. From the corresponding figure fig. 5.2, it can be seen that the training loss drops drastically after the first epoch and then remains the similar level. For this model, it has already learned most of the information that it is able to learn in the first epoch. As a result, the initial testing loss is not much higher than the final testing loss. From this behavior, it is assumed that model Linear_F2_L2 cannot learn to reduce the dimension well.

Putting aside model Linear_F2_L2, all initial and final loss of the training loss of linear models are smaller than the LSTM models, it is therefore assumed that the LSTM models don't perform significantly better than linear models.

Comparing between the choice of latent size among model with the same same feature size and same type (linear or LSTM), it can be observed that reduction percentage gets smaller when the latent size increases. It is assumed that models with larger latent size can pick up more information in the very first epoch. For models with same type and same latent size, the reduction percentage increase with the feature size but not significantly higher.

6.1.2 Reconstruction Result Analysis

Ideally, the reconstructed data should be identical as the original data. Comparing the reconstructed and the original data is a means of evaluating the performance of a model. Fig. 5.14 shows that the loss decreases when latent size is enlarged, and LSTM models perform better than linear models in most cases. Furthermore, increasing feature size does not improve the overall reconstruction loss.

From the figures in sec. 5.1.2, it can be seen that most reconstructed data are already close to the original data. However, the model Linear_F2_L2 cannot reconstruct the displacement data, which confirms the assumption in sec. 6.1.1 that this model cannot function well. In addition, model Linear_F3_L3 also cannot perform well and fail to reconstruct the velocity data. On the other hand, LSTM_F2_L2 model can reconstruct the displacement data, and although the LSTM_F3_L3 model also cannot reconstruct the velocity data as good as other models, but it is still better than the Linear_F3_L3 model. Moreover, compared with valve status and displacement, velocity is more difficult to reconstruct, and this can explain why 3-feature models have higher loss, because they need to reconstruct velocity data while 2-feature models do not.

Overall, when latent size is too small and both the linear model and LSTM models cannot learn well, but the LSTM models perform significantly better. When latent size is greater or equal to 4, bot linear and LSTM models perform well, but LSTM models still have advantages over linear models. However, the time required for training LSTM models are significantly longer than linear models, as shown in fig. 5.1. This is the trade-off that researchers have to consider.

6.1.3 Classification Result Analysis

The ideal classification result is that the accuracy equals to 100, meaning the labels of the testing dataset are 100% correctly predicted. The results in sec. 5.1.3 are in general very positive. Most models can reach an accuracy over 90% except for the model Linear_F2_L2 and LSTM_F2_L2, which confirms the assumption in sec.6.1.2 that latent size 2 may not be enough for our application.

Interestingly, although model Linear_F3_L3 fail to reconstruct the velocity data, it can still make good prediction on the label of dataset. It is therefore assumed that velocity data may not be important for the classification task.

Comparing the two different method of organizing the training data for classifiers, classifying per sequence (algo. 3) is obviously a better choice. Excluding model Linear_F2_L2 and LSTM_F2_L2, the effect of adjusting the feature size and latent size are not significant.

6.1.4 Latent Values Analysis

Latent values can be viewed as compressed input data, and are expected to contain critical information. Therefore, I also investigated deeper on latent values and the results are shown in 5.1.4.

Firstly, statistics of latent representations are calculated and presented. It can be seen that the latent sets resulted from each AE model are different to each other, but more information cannot be intuitively deduced from the statistics numbers.

Secondly, latent values of different phases of a data sequence are compared between each other. The distribution of averaged values are different in each phase of the data, meaning the information of the original data is indeed encoded in the latent values.

Lastly, the latent set of different leakage categories are compared between each other. For all AE models, both two external leakage categories are distinctly different from others. However, in two dimensional latent models, there is an overlap between no leakage and internal leakage category. For Linear_F2_L4 and Linear_F2_L4 , the overlap is reduced but the boarder between no leakage and internal leakage is still not clear. For other $3-$feature models, the 4 leakage categories are in 4 obvious clusters, meaning the AE models are able to compress the input data well, and therefore result in high accuracy in the classification task.

6.2 Variational Autoencoder

The VAE models built in this research fail to perform the reconstruction task. The results are shown in sec. 5.2. The self-consistent VAE models show higher uncertainty than basic VAE models, but both models cannot reconstruct the input data, therefore it cannot be put whether the basic VAE models are better or not. The models learn to output numbers close to a certain value, instead of learning the pattern of the whole data sequence. During the training process, what the models do is simply adjusting that specific target value of output. The potential reasons behind the failure of these VAE models can be:

These VAE models are too complex for our simple dataset. VAE can be applied to advanced tasks such as image processing, but is usually not used on simple tasks, since there are usually less complex algorithms for less complex goals. For example, in this case, AE models are simpler than VAE and can already perform the task well. Another possibility is that the algorithm VAE itself is suitable for this dataset, but the architecture of the encoder and decoder are too complex.

Another hypothesis is that the standard normal distribution assumption of VAE's latent space is not suitable for this dataset. From the latent values created by the AE models as shown in sec. 5.1.4, the distributions of latent values are far from standard normal distribution. From observation, these distributions have multiple peaks, therefore maybe distributions such as Mixture of Gaussians would

describe the latent space better. However, this is not in the scope of this research, but can be further investigated in future works.

7 Conclusion

This thesis shows the possibility of reducing the size of data from a fluid power system to low dimensional data representations (0.5% to 16% of the size of original data), but this data representation holds enough amount of information for further condition monitoring studies. The deep AE models are able to compress high-dimensional time-series data to separate regions in the latent space for different leakage statuses. The distinct separation paves the way for the leakage status classifiers, which are trained by the reduced dataset, to achieve high accuracy. In addition, the model training pipeline proposed by this thesis can also be adapted to data collected from other machines, and the time-series data are not required to have the same sequence length, which makes this pipeline more general for different applications.

The potential directions that future researches can focus on are listed as:

1. Investigate on variational autoencoder models

2. Take experimental data as input

Although AE models are implemented in this thesis are already powerful enough to encode information that can be used to identify leakage status, VAE models can bring a useful benefit: the decoder of VAE can be used to generate artificial data. The latent space in a VAE model is regulated by the KL loss, making the latent space continuous. For an AE model, if a random value between two encoded latent values is fed to the decoder, the output may not be meaningful. However, for a VAE model, this output will contain characteristics of both the two existing latent values. For example, if an artificial data sequence of both external rod leakage and external piston leakage happening at the same time is desired, it may be achieved by sampling a point from the latent space which is positioned between the region of external rod leakage and external piston leakage. This property of VAE can be helpful in providing a simulated dataset. Another topic that can be carried out in future works is to take real data as input instead of simulated data. Experimental data contain more noise, and the stability of the proposed models and pipeline can be therefore tested.

Overall, in this thesis, a technique of condition monitoring of a fluid power system using deep unsupervised learning and trained by a reduced variable time period dataset is presented. Moreover, the implemented framework establishes a pipeline useful for further research.

List of Figures

List of Tables

References

[1] N. Helwig, E. Pignanelli, and A. Schütze, "Condition monitoring of a complex hydraulic system using multivariate statistics," in *2015 IEEE International Instrumentation and Measurement Technology Conference (I2MTC) Proceedings*, 2015, pp. 210–215.

[2] R. Isermann, "Fault-diagnosis applications: Model-based condition monitoring: Actuators, drives, machinery, plants, sensors, and fault-tolerant systems," 2011.

[3] F. Makansi and K. Schmitz, "Simulation-based data sampling for condition monitoring of fluid power drives," *IOP Conference Series: Materials Science and Engineering*, vol. 1097, 2021.

[4] T. Kovacs and A. Ko, "Monitoring pneumatic actuators' behavior using real-world data set," *SN Computer Science*, vol. 1, 06 2020.

[5] M. Karpenko and N. Sepehri, "Neural network classifiers applied to condition monitoring of a pneumatic process valve actuator," *Engineering Applications of Artificial Intelligence*, vol. 15, pp. 273–283, 06 2002.

[6] M. Demetgul, I. Tansel, and S. Taskin, "Fault diagnosis of pneumatic systems with artificial neural network algorithms," *Expert Systems with Applications*, vol. 36, no. 7, pp. 10 512–10 519, 2009. https://www.sciencedirect.com/science/article/pii/S0957417409000657

[7] H. Sano, J. P. P. Malere, and L. Berton, "Single and multiple failures diagnostics of pneumatic valves using machine learning," 10 2019.

[8] C. Lurette and S. Lecoeuche, "Unsupervised and auto-adaptive neural architecture for on-line monitoring. application to a hydraulic process," *Engineering Applications of Artificial Intelligence*, vol. 16, pp. 441–451, 08 2003.

[9] M. Münchhof, "Model-based fault detection for a hydraulic servo axis," in *Autom.*, 2007.

[10] A. Zachrison, "Fluid power applications using self-organising maps in condition monitoring," 01 2008.

[11] Z. Wang, C. Lu, H. Liu, and z. Chen, "Performance assessment of hydraulic servo system based on radial basis function neural network and mahalanobis distance," *Applied Mechanics and Materials*, vol. 764-765, pp. 613–618, 05 2015.

[12] L. Siyuan, D. Linlin, and J. Wanlu, "Study on application of principal component analysis to fault detection in hydraulic pump," 08 2011.

[13] M. Geimer and P.-M. Synek, Eds., *11. Kolloquium Mobilhydraulik : Karlsruhe, 10. September 2020*, ser. Karlsruher Schriftenreihe Fahrzeugsystemtechnik / Institut für Fahrzeugsystemtechnik, vol. 83. KIT Scientific Publishing, 2020.

[14] S. Chawathe, "Condition monitoring of hydraulic systems by classifying sensor data streams," 01 2019, pp. 0898–0904.

[15] J. R and V. Sugumaran, "Fault diagnosis of automobile hydraulic brake system using statistical features and support vector machines," *Mechanical Systems and Signal Processing*, vol. 52, 02 2015.

[16] N. Helwig, E. Pingnalli, and A. Schütze, "Detecting and compensating sensor faults in a hydraulic condition monitoring system," 05 2015.

[17] T. Schneider, N. Helwig, and A. Schütze, "Automatic feature extraction and selection for classification of cyclical time series data," *tm - Technisches Messen*, vol. 84, no. 3, pp. 198–206, 2017. https://doi.org/10.1515/teme-2016-0072

[18] Y. Lei, F. Jia, J. Lin, S. Xing, and S. Ding, "An intelligent fault diagnosis method using unsupervised feature learning towards mechanical big data," *IEEE Transactions on Industrial Electronics*, vol. 63, pp. 1–1, 05 2016.

[19] T. Krogerus, J. Vilenius, J. Liimatainen, and K. Koskinen, "Self-organizing maps with unsupervised learning for condition monitoring of fluid power systems," *SAE Technical Papers*, 10 2006.

[20] T. Krogerus, *Feature Extraction and Self-organizing Maps in Condition Monitoring of Hydraulic Systems*. Tampereen teknillinen yliopisto., 2011. https://books.google.de/books?id=bFwIvwEACAAJ

[21] Z. Huijie, R. Ting, W. Xinqing, Z. L. You, and F. Husheng, "Fault diagnosis of hydraulic pump based on stacked autoencoders," *2015 12th IEEE International Conference on Electronic Measurement & Instruments (ICEMI)*, vol. 01, pp. 58–62, 2015.

[22] A. Mallak and M. Fathi, "Sensor and component fault detection and diagnosis for hydraulic machinery integrating lstm autoencoder detector and diagnostic classifiers," *Sensors*, vol. 21, no. 2, 2021. https://www.mdpi.com/1424-8220/21/2/433

[23] J. Hennig, A. Umakantha, and R. Williamson, "Sequence generation and classification with vaes and rnns," in *Proceedings of the 34 th International Conference on Machine Learning*. Sydney, Australia: JMLR: WCP, 2017.

[24] B. Schiele, "Lecture notes in elements of data science and artificial intelligence," November 2019.

[25] S. Marsland, *Machine Learning: An Algorithmic Perspective 2ed, Chapman Hall*, 2014.

[26] M. K. V. S. Anas Abdelrazeq, Andrés Posada, "Lecture 6, artificial intelligence and data analytics for engineers," June 2020.

[27] T. Yiu. (2019) Understanding random forest. https://towardsdatascience.com/understanding-random-forest-58381e0602d2

[28] C. Nicholson. (2020) Recurrent neural networks and lstm explained. https://wiki.pathmind.com/multilayer-perceptron

[29] R. Bellman, *Adaptive Control Processes, Princeton University Press, Princeton, NJ*, 1961.

[30] J. C. Ian T. Jolliffe, "Principal component analysis: a review and recent developments," *Philosophical Transactions of the Royal Society A: Mathematical, Physical, and Engineering Sciences*, April 2016. https://doi.org/10.1098/rsta.2015.0202

[31] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," *Workshop on Unsupervised and Transfer Learning*, pp. 27:37–50, 2012.

[32] M. K. V. S. Anas Abdelrazeq, Andrés Posada, "Lecture 8, artificial intelligence and data analytics for engineers," June 2020.

[33] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2014.

[34] F. Rashad. (2020) Generative modeling with variational auto encoder (vae). https://medium.com/vitrox-publication/generative-modeling-with-variational-auto-encoder-vae-fc449be9890e

[35] M. Patacchiola. (2021) Evidence, kl-divergence, and elbo. https://mpatacchiola.github.io/blog/2021/01/25/intro-variational-inference.html

[36] E. Norlander and A. Sopasakis, "Latent space conditioning for improved classification and anomaly detection," 2019.

[37] F. Rosenblatt, *The Perceptron, a Perceiving and Recognizing Automaton*, ser. Report: Cornell Aeronautical Laboratory. Cornell Aeronautical Laboratory, 1957. https://books.google.de/books?id=P_XGPgAACAAJ

[38] P. Gudikandula. (2019) Recurrent neural networks and lstm explained. https://purnasaigudikandula.medium.com/recurrent-neural-networks-and-lstm-explained-7f51c7f6bbb9

[39] R. Wei, C. Garcia, A. El-Sayed, V. Peterson, and A. Mahmood, "Variations in variational autoencoders - a comparative evaluation," *IEEE Access*, vol. 8, pp. 153 651–153 670, 2020.

[40] J. D. Co-Reyes, Y. Liu, A. Gupta, B. Eysenbach, P. Abbeel, and S. Levine, "Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings," 2018.

[41] I. Higgins, L. Matthey, A. Pal, C. P. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "beta-vae: Learning basic visual concepts with a constrained variational framework," in *ICLR*, 2017.

[42] K. Gregor, G. Papamakarios, F. Besse, L. Buesing, and T. Weber, "Temporal difference variational auto-encoder," in *International Conference on Learning Representations*, 2019. https://openreview.net/forum?id=S1x4ghC9tQ