Representation learning for ball nut assembly signals by time delayed embeddings, variational autoencoders and mutual information maximization

Yu He
Matriculation number: 5125263
yu.he@fu-berlin.de

Betreuer: Prof. Dr. Daniel Göhring
Zweitkorrektor: PD Dr. Pavel Gurevich
Thesis date: Friday 23rd November, 2018

Masterarbeit am Institut für Informatik der Freien Universität Berlin

Abstract

Representation learning, a set of techniques of automatically discovering the useful features for further processing, significantly influences the final results of the task. However, most of current representation learning methods cannot be applied on the time series data due to the temporal nature of the time series. In order to extract features from this type of date, we propose three representation learning methods. The first one is based on time delayed embeddings, Wasserstein distance, and multidimensional scaling. The second method is based on the variational autoencoder, which is a powerful deep learning model that learns the representation of the data and even generates data of a similar pattern. The third method learns the representation efficiently based on mutual information, which belongs to the field of semi-supervised learning. In this thesis, we first explain these three methods mathematically and use examples to illustrate them. Then we implement these methods and test them using standard dataset. Finally, we train these models using the time series data from ThyssenKrupp and demonstrate that the representations given by these three methods share certain consistency. In addition, this master thesis is based on the project with ThyssenKrupp. The time series data is also generated by the rotation of the products produced by ThyssenKrupp.

Selbständigkeitserklärung

Ich erkläre gegenüber der Freien Universität Berlin, dass ich die vorliegende Masterarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe.

Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht. Diese Arbeit wurde in gleicher oder ähnlicher Form noch bei keiner anderen Universität als Prüfungsleistung eingereicht.

Datum: Friday 23rd November, 2018

Name: Yu He

Sperrvermerk

Die vorliegende Masterarbeit mit dem Titel:

Representation learning for ball nut assembly signals by time delayed embeddings, variational autoencoders and mutual information maximization

beinhaltet interne und vertrauliche Informationen und Daten des Unternehmens:
thyssenkrupp Presta AG
Eine Einsicht in diese Masterarbeitarbeit ist ohne ausdrückliche Zustimmung des Unternehmens thyssenkrupp Presta AG und des Verfassers nicht gestattet. Ausgenommen davon sind die betreuenden Dozenten, Gutachter sowie die befugten Mitglieder des Prüfungsausschusses. Eine Veröffentlichung und Vervielfältigung der Masterarbeit – auch in Auszügen – ist nicht gestattet.
Ausnahmen von dieser Regelung bedürfen einer Genehmigung des Unternehmens thyssenkrupp Presta AG.

Contents

1 Introduction **1**

2 Background **3**

2.1 Task Statement 3

2.2 Related Works 4

3 Time delay embedding based method **6**

3.1 Time delayed embedding 6

3.1.1 Definition 6

3.1.2 The choice of parameters 7

3.1.3 Example: Attractor reconstruction using time delayed embedding 8

3.2 Further processing steps 10

3.2.1 Probability measure interpretation and Wasserstein distance 10

3.2.2 Multidimensional scaling 12

3.2.3 Multivariate Gaussian fitting and neural network for regression 13

3.3 Frequency spectrogram with Wasserstein distance and MDS as alternative 13

4 Variational autoencoder based method **16**

4.1 Preliminaries: Deterministic autoencoders and latent variables 16

4.2 Derivation of loss function in variational autoencoders 17

4.2.1 Closed-form KL divergence 19

4.2.2 Reformulation of the reconstruction loss 20
4.3 Reparameterization trick in VAE 20

5 Squared-loss mutual information based method **23**
5.1 Mutual information and problem definition 23
5.2 SMI derivation and approximation 25
5.3 Representation learning and classification using SMI 26

6 Experiments and evaluation **28**
6.1 Experiments using standard datasets 28
6.1.1 Time delayed embedding method 28
6.1.2 VAE method 30
6.1.3 SMI method 35
6.2 Experiments using ThyssenKrupp dataset 39
6.2.1 General preprocessing of data 40
6.2.2 Time-delay-embedding method 41
6.2.3 VAE method 45
6.2.4 SMI method 45
6.3 Consistency in different results 46

7 Summary and further works **52**

Bibliography **56**

List of Figures

2.1 One BNA example 3
2.2 One example time series from ThyssenKrupp 4

3.1 One damped pendulum system 8
3.2 Time series of x, y and z 9
3.3 Lorenz attractor and reconstruction 9
3.5 One example of optimal transportation problem 12
3.6 real Fourier transform examples 14

4.1 One example of autoencoder architecture 17
4.2 VAE with and without reparameterization trick 21

5.1 Relationship between BNA and final product 24
5.2 An overview of the neural network for SMI method 27

6.1 Example time series and spectrogram of bass and vocal audio 29
6.2 Example digits from MNIST 29
6.3 Example time blocks generated from bass and vocal audio 30
6.4 Learned representation based on time domain using bass and vocal audio 31
6.5 Example input from bass and vocal audio for Wasserstein distance calculation . . 32
6.6 Learned representation based on frequency domain using bass and vocal audio . 32
6.7 Learned representation of test set using MNIST dataset 33

6.8 Generated digits using MNIST dataset 34
6.9 Learned representation of test set with $\theta_P = 0.3$ using MNIST dataset 36
6.10 Learned representation of test set with $\theta_P = 0.5$ using MNIST dataset 38
6.11 Learned representation of test set with $\theta_P = 0.7$ using MNIST dataset 39
6.12 One original time series in ThyssenKrupp dataset 40
6.13 Sub time series generated by splitting the original from ThyssenKrupp dataset . 41
6.14 Example of original and convoluted spectrogram based on ThyssenKrupp data . 41
6.15 Learned representation of ThyssenKrupp data based on time domain 43
6.16 Proportion of black points from periphery to center based on time domain using ThyssenKrupp dataset 44
6.17 Learned representation of ThyssenKrupp data based on frequency domain 45
6.18 Proportion of black points from periphery to center based on frequency domain using ThyssenKrupp dataset 46
6.19 Spectrograms of sampled points at center and at periphery using ThyssenKrupp dataset 47
6.20 Learned representation of ThyssenKrupp data based on frequency domain using L_2 distance 48
6.21 Representation of convoluted frequency data from ThyssenKrupp using VAE . . 49
6.22 Learned representation of convoluted frequency data from ThyssenKrupp using SMI 50
6.23 Consistency of reprensetations in different methods using ThyssenKrupp dataset 51

List of Tables

6.1 Number and rate of misclassification with $\theta_P = 0.3$ using MNIST dataset 37
6.2 Number and rate of misclassification with $\theta_P = 0.5$ using MNIST dataset 37
6.3 Number and rate of misclassification with $\theta_P = 0.7$ using MNIST dataset 40

Chapter 1

Introduction

Machine learning and deep learning nowadays have broad applications in industry. Because industry companies have been using sensors to collect data during the producing or testing procedure for many years. A large amount of data is the cornerstone of deep learning. The development of central processing units (CPUs) and graphical processing units (GPUs) helps to accelerate the learning process. However, the learning algorithms are heavily dependent on the features of the data [1] and the raw data from sensors cannot directly support learning algorithms. The process of raw data reconstruction is needed, which is called feature engineering. If this process is manually done, it requires the expertise of the corresponding domain and is also time-consuming. Hence, many representation learning algorithms are proposed. For example, Principle Component Analysis (PCA) is a representation learning method based on unsupervised learning. It projects data from high-dimensional space to low-dimensional space and tries to remain useful information as much as possible. The result in lower dimensional space is the reconstructed representation and is the input for further classification or regression. The reduced dimensionality helps the classifier or the predictor to improve the efficiency and accuracy.

However, most of the methods from representation learning community focus on developing models for static data instead of time series data [16]. Time series data consists of samples in a time order. In [16], the authors conclude that time series data is high-dimensional and complex with unique properties that make them challenging to analyze and model.

In this paper, we propose three types of representation learning methods for one specific type of time series data given by ThyssenKrupp. The first method is based on time delayed embeddings, which is a method based on Takens's theorem in dynamical system analysis. In this method, we assume that the time series is generated by a dynamical system. Firstly, the attractor of the time series is reconstructed based on time delayed embeddings. We then asso-

ciate probability measures to the attractors. Afterward, we calculate the Wasserstein distance between each probability measure and obtain a Wasserstein distance matrix describing the dissimilarity between each time series. Through Multidimensional Scaling (MDS), we obtain coordinates in m dimensional space using distance matrix from the previous step. Further processing includes multivariate Gaussian fitting to obtain probability density and training neural networks for regression. In order to compare with our time delayed embeddings using time domain information, we also use Fourier transform to analyze the data in the frequency domain. The measure is constructed based on the result of the Fourier transform, which is the spectrogram. The following procedures are the same as the steps of the time delayed embedding method.

The second method is based on variational autoencoder (VAE), which is a generative deep learning model that has made remarkable progress in recent years. This model is based on Bayesian inference and can model fairly complex datasets, including natural images and speech. The general idea is that it uses the probabilistic encoder to encode the input to probabilistic distributions in latent space. The samples from these distributions are decoded by a probabilistic decoder to regenerate the input. The performance of the encoder and the decoder, therefore, can be measured.

The third representation learning method is based on mutual information. The entire algorithm simultaneously learns the representation and produces the classification result based on the learned representation.

The rest of this thesis is organized as follows. Section 2 gives detailed descriptions of the data and defines the task given by ThyssenKrupp. The related works are also mentioned. In section 3, 4, and 5, we describe the methods and the mathematical reasoning in detail. We also use simplified examples for better explanations. Section 6 shows how the proposed methods are applied to the standard dataset MNIST, standard dataset NSynth, and ThyssenKrupp dataset. Finally, we discuss the limitations of the methods and potential future directions in section 7.

Chapter 2

Background

2.1 Task Statement

As mentioned in the introduction part, the time series data is given from ThyssenKrupp. It is generated by rotating one product called ball screw or ball nut assembly (BNA) produced in ThyssenKrupp factories in Germany. Figure 2.1 is one example of BNA. It is a mechanical linear actuator that translates the rotational motion to the linear motion with little friction. Low friction in ball screws yields high mechanical efficiency compared to conventional lead screws.

Figure 2.1: One BNA example

The BNAs made in Germany are shipped to other countries *e.g.* Mexico, France and China. They are assembled in these countries into the final product, steering gear. The steering gear is composed of different small components. One of the small components is the BNA. If the final product steering gear satisfies the consumers' requirement, then all small components are good. If not, then at least one small component is the defect.

To know whether it is because of the BNA, they are put into a rotation machine and rotated at a constant speed. The sensor in the rotation machine records this process with a sample rate of 25.6 kHz. One example of the time series is shown in figure 2.2. The labels of the final products are given. It is either defect or good.

To summarize, we have time series generated by rotating BNA and the labels representing

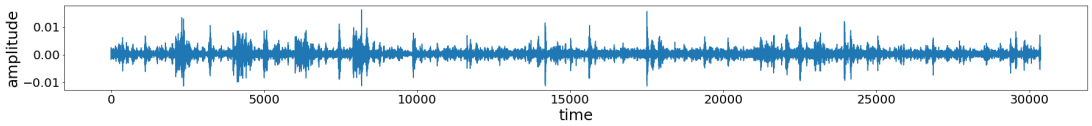


Figure 2.2: One example time series from ThyssenKrupp

the quality of the final products, steering gears.

The task is to find a representation of these time series, which shows the quality of the BNA according to the labels of final products.

2.2 Related Works

Among time series representation learning methods, a family of methods is based on deep learning. The representation learning happens inside the deep hidden layers. Convolutional Neural Networks (CNNs) use filters to do convolution on the input time series. The values in the filters are obtained after the training process. The result of convolution, also called receptive field, is the new representation extracted by filters. It is the input of the fully connected layers for further classification or regression. In [28], the author implemented CNNs to recognize the human activities based on the data from multiple body-worn sensors.

Recently, another type of neural network, Recurrent Neural Networks (RNNs), a class of neural networks where connections between nodes form a directed graph along a sequence, have been considered as an effective model for time series analysis. It directly takes the data at each time step as inputs. The difference between RNNs and traditional feed-forward neural networks is that the output of the hidden layers is not only given to the next layer but also reused as input together with the data of the next time step. It takes the temporal features into consideration during the training process. The predictions from the output layer are based on the outputs of previous hidden layers, which are the new representations that we desire. One main problem for using RNNs is the bad performance while long-time dependency exists in the data. In [21], authors explain that the bad performance is caused by vanishing or exploding gradient during the training process. Furthermore, in [2], authors claim that learning long-term dependencies with gradient descent is difficult.

Its variant, Long-Short-Term-Memory (LSTM), has solved the vanishing or exploding gradient problem in RNNs by introducing cell state and other 3 gate units. The representation learning in this kind of neural networks is embedded into the hidden layers. LSTM now plays an important role in many real-world applications like language modeling [26], voice recognition [22].

Despite the success achieved by RNN and LSTM, the properties of this kind of neural networks are still not clear, which makes the hyperparameters tuning tricky. [13] designs experiments during testing to check the gate activation statistics to understand the mechanism. [10] analyzes the hyperparameter interactions. The analysis results imply that for practical purposes the hyperparameters can be treated as approximately independent. At last, authors concluded that " this has remained a hurdle for newcomers to the field since a lot of practical choices are based on the intuitions of experts, as well as experiences gained over time."

There are also methods, which are not based on deep learning. For instance, authors of [20] proposed to use the statistical features including means, standard deviations, kurtosis, and skewness from the original time series to form a feature vector for further processing. There are also methods proposed to use the result of Fourier transform [8] or wavelet transform [5] as the new representation.

Despite the remarkable progress of representation learning for time series, the neural network based techniques lack the interpretability. During the implementation, fine-tuning of the hyperparameters is a crucial and tricky step. For the methods without deep learning, they are usually domain-specific and cannot be generalized to be applied to many other domains.

Chapter 3

Time delay embedding based method

3.1 Time delayed embedding

3.1.1 Definition

As described in 2.1, the time series we have is generated by the rotation of BNA at a constant speed. Here we assume that each time series is generated by some certain dynamical system. At any given time, a dynamical system has a state given by a tuple of real numbers (a vector) that can be represented by a point in an appropriate state space [9]. The evolution rule of the dynamical system is a system of ordinary differential equations (ODEs) that describe what future states follow from the current state [24]. Nearly every nontrivial real-world system is a nonlinear dynamical system.

The repeated measurements correspond to a multidimensional dynamical system. In our case, the measurements are the data recorded by the sensor in the rotation machine. The measurements are the projections of the trajectories of this dynamical system. And time delayed embeddings method allows us to reconstruct the attractor of the original trajectories.

Assume we can observe the time series $x = [x_1, x_2, \cdots x_t]$[1] and each element x_i in the time series is a measurement at time step i. The formula of time delayed embedding on time series x is:

$$x_{[i]} = (x_i, x_{i+q}, x_{i+2q} \cdots x_{i+(k-1)q}) \in \Omega := \mathbb{R}^k \tag{3.1}$$

We call $x_{[i]}$ a block. Each block is a k dimensional point in Euclidean space $\mathbb{R}^k$. There are

[1] We use square brackets to represent time series

three parameters controlling the time delayed embedding, the starting time step of sampling i, the time lag of sampling q, and the number of samplings in each block k. If we continually select different i values while keeping time lag q and the number of dimensions k fixed, then we obtain a discrete trajectory in $\mathbb{R}^k$.

From a statistical perspective, a block captures the higher-order correlations [19]. From a dynamical system perspective, this way of sampling and reconstruction from equation (3.1) is an embedding and it contains the information at time step i and also the information after time step i [19].

3.1.2 The choice of parameters

Firstly, for the value of time lag q, it is arbitrary since the data items are assumed to have infinite precision from a mathematical point of view. Also, it is assumed that there is an infinite number of them [12]. But the situation in applications is different. A proper time lag q is crucial for results. If the value of q is too small, then each block $x_{[i]}$ might be almost the same as the original time series. This is called redundancy in [4]. A small time lag might increase the influences of noise in the block, too. However, if q is too large, then each block may be no more correlated. In this case, the trajectory becomes complicated, even if the true one is simple. This is typical of chaotic systems [12]. As authors of [12] suggested, in practice, we use different many q values and find the q value with the best performance. But the results should not be too dependent on the value of q.

As for the number of dimensions k, Takens could prove that it is a generic property that a delayed map of dimension $k = 2D+1$ is an embedding [27], where D is the integer dimension of a smooth manifold. Since this thesis is about the real-world application but not mathematical concepts. The idea here to know is that the value of D can be much larger than the measurement dimension [12]. Similar to the principle of choosing the value of time lag q, the results should not depend on k too much.

The value of the starting sampling point i depends on the two previous parameters' values, after deciding the value of time lag q and the number of dimensions k, we make sure that under the current values of is, each block doesn't overlap with each other. The overlapped block contains redundant information, which is meaningless for further processing.

In summary, time delayed embedding gives us a mapping from the original time series dimension to another dimension in Euclidean space. This method requires 3 parameters i, q, k that need to be defined manually. Once the latter two are decided, the first one is easy to choose. And the results should not depend too much on these parameters.

3.1.3 Example: Attractor reconstruction using time delayed embedding

Time delayed embedding can be applied on the attractors. An attractor is a value or a set of values that a system settles toward over time [3]. A rough definition of it is the set of all recurrent states of the system. One example of an attractor is the minimum height x_1 of the ball in a damped pendulum system. As figure 3.1 shows, the x_1 is minimum height position. Because this is a damped system and there exists the friction from the pivot and the friction in the air, the ball stops finally at the position x_1. x_1 is an attractor in this system.

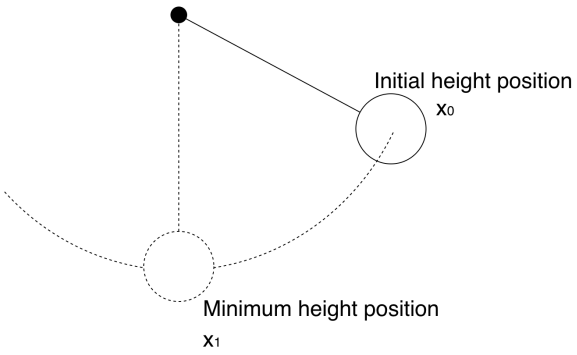


Figure 3.1: One damped pendulum system

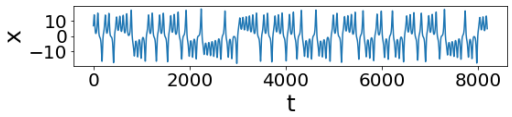
Another more complicated example of the attractor is the Lorenz attractor. Lorenz attractor is defined by the following ODEs below:

$$\begin{aligned}\frac{dx}{dt} &= a(y-x)\\ \frac{dy}{dt} &= x(b-z)-y\\ \frac{dz}{dt} &= xy-cz\end{aligned}$$

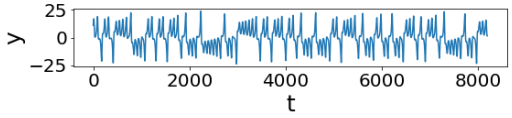
where $a = 10, b = 28, c = \frac{8}{3}$.

Given the initial state of x_0, y_0, z_0, and time step unit t, we can obtain x_1, y_1, and z_1 values. Continually doing this, we have the entire time series. It is shown in figure 3.2.

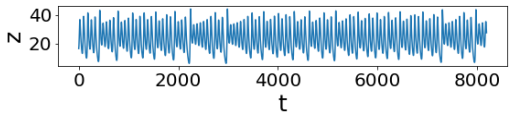
Each (x_i, y_i, z_i) is a point in $3D$ space, as figure 3.3a shows. This is known as the Lorenz attractor. Assume we only observe the time series x and we want to know the underlying dynamical system. We use time delayed embedding defined by equation (3.1) for the time



(a) Time series x



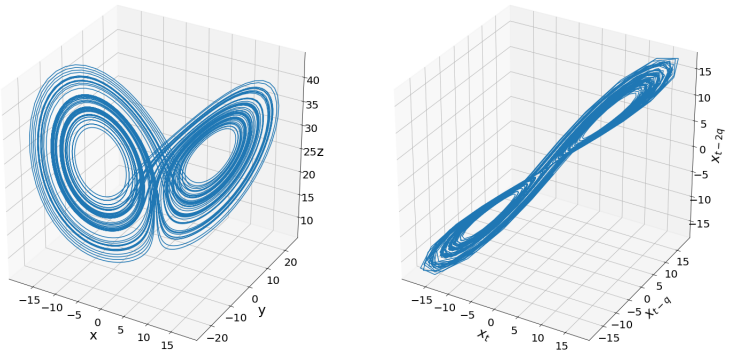
(b) Time series y



(c) Time series z

Figure 3.2: Time series of x, y and z

series x with $i = 0, 2, 4, 6 \ldots$, $q = 2$, and $k = 3$. Then we have a set of blocks $x_{[i]}$. Each block $x_{[i]}$ is a point in the reconstructed space $\Omega = \mathbb{R}^3$. The trajectory $\{x_{[i]}\}_{i=0,2,4\ldots}$ is called a reconstructed attractor. The visualization is shown in figure 3.3b. The reconstructed attractor keeps topological features of the original attractor, comparing with the figure 3.3a. Without observing time series y and z, we have inspection of the underlying dynamical system. This example shows the effectiveness of time delayed embedding.



(a) Lorenz attractor (b) Reconstructed attractor

Figure 3.3: Lorenz attractor and reconstruction

We consider that the time series from ThyssenKrupp is the only observation that describes

the relationship between the time and the amplitude about the dynamical system related with the quality of the BNA. By using this method with the proper parameter values, the delayed embedding is a reconstruction of the dynamical system and gives us inspection.

3.2 Further processing steps

The first step after time delayed embedding is to interpret the results to the probability measures. So the outputs of this step are probability measures, which correspond to the original time series. Then based on the obtained probability measures, we calculate Wasserstein distance, a measure of probability distributions' dissimilarity. And the result of this step is a Wasserstein distance matrix. Assume we have n time series, then the Wasserstein distance matrix D is a $n * n$ symmetric matrix.

The second step is to use Multidimensional scaling (MDS), a method from manifold learning, to construct m dimensional points in Euclidean space. Again we assume the number of time series is n, then the output of MDS is a coordinate matrix C with a shape of $n * m$. Usually, m is 2 or 3 for the purpose of a better visualization.

In the third step, we fit multivariate Gaussian on the coordinates we get. So each point has a probability density. Then a neural network is trained to predict the probability density value, using the original time series.

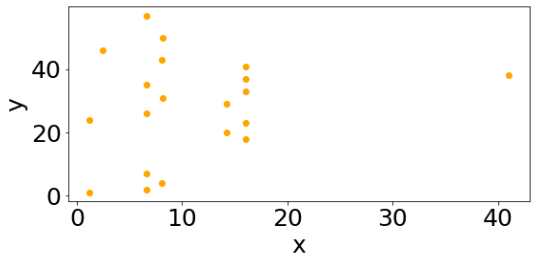
3.2.1 Probability measure interpretation and Wasserstein distance

Assume the result from the previous step has N blocks. According to equation (3.1), each block $x_{[i]}$ is a point in Euclidean space $\Omega \in \mathbb{R}^k$. We define a probability measure μ on each block using δ function as equation (3.2).

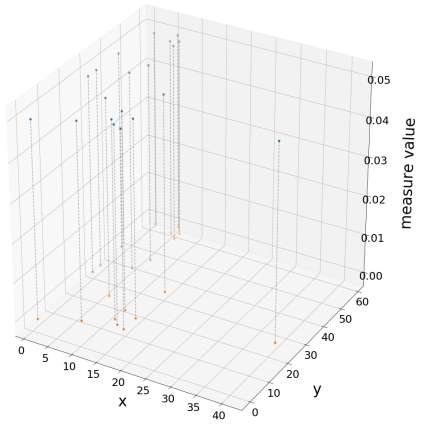
$$\mu[A] = \frac{1}{N} \sum_{i=1}^{N} \delta_{x_{[i]}}[A], \quad A \subseteq \Omega \tag{3.2}$$

For a better understanding of this interpretation, assume we have a set of points in $\mathbb{R}^2$. And it is generated from one time series using time delayed embedding. The number of these points is 20. The visualization of these points is shown in figure 3.4a. Based on the equation (3.2), each point obtains a probability of $\frac{1}{N}$, which is 0.05, since $N = 20$ in our case. Figure 3.4b shows the generated measure. The orange points on $x - y$ plane are the same as the points in figure 3.4a. And the blue points represent the corresponding measure value 0.05.

By doing this, each time series has a probability measure interpretation [19]. It allows us to measure the differences between two time series represented by two probability measures.



(a) One example spectrogram



(b) Generated probability measure

We choose Wasserstein distance to measure differences between two probability measures. It is also called earth mover distance (EMD). It is a distance function defined between probability distributions. The idea of Wasserstein distance is based on the optimal transportation problem (OPT).

Imagine we have one pile of dirt P and a hole Q. The volume of the dirt is the same as the volume of the hole. The cost of moving one unit of dirt to the hole is defined by the Euclidean distance between the location of the dirt and the destination of the dirt. There exists one way of transporting all the dirt from P to Q with minimal cost, which is called OPT. Figure 3.5 is one example scenario.

Now we change the dirt and the hole to two probability distributions P and Q, while other definitions remain unchanged. The minimum cost is defined as the Wasserstein distance between these two distributions.

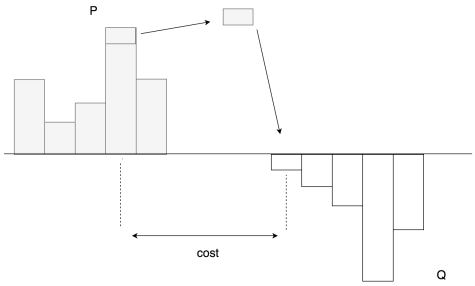


Figure 3.5: One example of optimal transportation problem

Mathematically formulation for Wasserstein distance is equation (3.3).

$$W(P,Q) = \inf_{\pi \in \Pi(P,Q)} \int_{\Omega \times \Omega} ||x-y||_2^2 d_{\pi[x,y]} \tag{3.3}$$

where Π is the set of all probability measures on the product $\Omega \times \Omega$ with marginals P and Q.

We calculate the Wasserstein distance between each probability measure. Suppose there are n time series, the result of this step is a Wasserstein distance matrix with a shape of $n * n$.

3.2.2 Multidimensional scaling

Using coordinates of the points to calculate the distance between them is straightforward. Assume point $x_i = (x_{i1}, x_{i2}, \cdots, x_{im})$ and $x_j = (x_{j1}, x_{j2}, \cdots, x_{jm})$. The squared Euclidean distance between x_1 and x_2 is calculated by equation (3.4).

$$d_{ij}^2 = \sum_{p=1}^{m} (x_{ip} - x_{jp})^2 \tag{3.4}$$

With more points, then the distances between each pair of the points form a distance matrix D, which is defined as equation (3.5).

$$D = \begin{bmatrix} 0 & d_{12}^2 & d_{13}^2 & \cdots & d_{1n}^2 \\ d_{21}^2 & 0 & d_{23}^2 & \cdots & d_{2n}^2 \\ \cdots & \cdots & \cdots & \cdots & \\ d_{n1}^2 & d_{n2}^2 & d_{n3}^2 & \cdots & 0 \end{bmatrix} \tag{3.5}$$

where each element $d_{i,j}^2$ in D means the squared distance between point i and j.

The inverse problem, from the distance matrix D to calculate the coordinate matrix, can

be solved by Multidimensional scaling (MDS). MDS takes the dissimilarity matrix, in our case distance matrix, and outputs the coordinate matrix that approximates the dissimilarity matrix as much as possible. The dimensionality m of coordinates is manually decided. Usually we choose $m = 2$ or $m = 3$ for an easier visualization. The validity of MDS for Euclidean distance matrix has been proved in [11].

After MDS is applied to the Wasserstein distance matrix obtained from the previous step, we have the coordinate matrix with a shape of $n*m$ representing n time series in m-dimensional space.

3.2.3 Multivariate Gaussian fitting and neural network for regression

The result from MDS is a point cloud. By observation of this point cloud, we approximately find the number of clusters in this point cloud. If the observation can't directly tell us, by using the K-means algorithm with different initial values of k and elbow plot, we obtain the optimal number of clusters [15]. Once the number of clusters is certain, we use Gaussian distribution to fit the point cloud. And each point in the point cloud obtains a probability density. The closer to the center of the cluster, the larger probability the point has.

Each point with a probability density in the point cloud is the final representation of the time series. But to learn this representation is time-consuming because of the Wasserstein distance calculation. Once we have a new time series and want to know where it locates in the point cloud, the Wasserstein distance matrix needs to get updated, which means that we need to re-calculate the Wasserstein distance for all the time series again. This is inefficient and a waste of time. So we train a neural network using the original time series as the inputs and the corresponding probability density as the target value. Now if we have a new time series and wonder where it could be in the point cloud. We feed the new time series to the well-trained neural network and get the corresponding prediction of probability density. Based on the prediction, we know where the point possibly locates in the point cloud.

3.3 Frequency spectrogram with Wasserstein distance and MDS as alternative

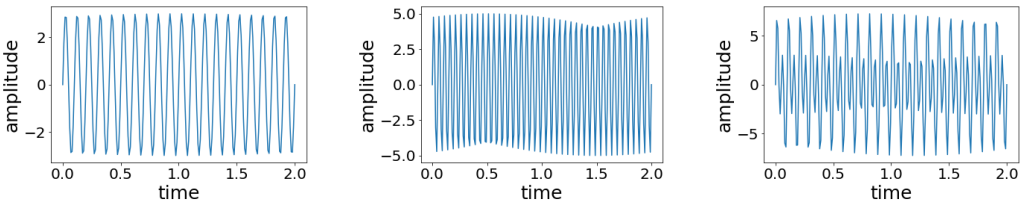
The time delayed embedding is based on the information from the time domain. As an alternative approach, we use Fourier transform on the time series to obtain the information from the frequency domain. Fourier transform is a method for expressing a function as a sum of periodic components. It allows us to transform the original time series into spectrogram in the frequency domain. Since our input time series is discrete and purely real, only the positive

frequency components are computed. This transform is the real Fourier transform

One example of the real Fourier transform on the sum of cos waves is shown in figure 3.6. There are two discrete cos waves $\tilde{x}_1$ and $\tilde{x}_2$ sampled from the continuous cos wave x_1 and x_2 with a sample frequency of $f_s = 100$, respectively. A_i is the amplitude of cos wave x_i and the f_i represents the frequency of x_i. The sum of $\tilde{x}_1$ and $\tilde{x}_2$ is $\tilde{x}_3$.

$$x_1 = A_1 \cos(2\pi f_1 x)$$
$$x_2 = A_2 \cos(2\pi f_2 x)$$
$$x_3 = x_1 + x_2$$

where $A_1 = 3, f_1 = 10$ and $A_2 = 5, f_2 = 20$



(a) Sampled cos wave $\tilde{x}_1$ (b) Sampled cos wave $\tilde{x}_2$ (c) Sampled cos wave $\tilde{x}_3$

(d) Spectrogram of $\tilde{x}_1$ (e) Spectrogram of $\tilde{x}_2$ (f) Spectrogram of $\tilde{x}_3$

Figure 3.6: real Fourier transform examples

As figure 3.6d and figure 3.6e show, the spikes in each plot are at the corresponding frequency f_1 in x_1 and f_2 in x_2. The spike in figure 3.6f is the sum of the spikes in figure 3.6d and 3.6e.

In summary, the output of real Fourier transform is the spectrogram, which is composed of 2-dimensional points. One dimension represents the frequency and another is the corresponding amplitude. The number of these points N' is decided by the Nyquist sampling theorem, which is equation (3.6).

$$N' = \frac{f_s}{2} \tag{3.6}$$

where f_s is the sampling frequency.

The following steps are the same as those used in time delayed embeddings, including

defining probability measure, calculating the Wasserstein distance between each measure, and using MDS for obtaining m dimensional representation.

One remark is that the result from time delayed embedding is a set of k dimensional points, which perfectly fits the situation to use measure function then calculate Wasserstein distance since each time series is represented by a set. However, if the result is from real Fourier transform, it also makes sense to use L_2 distance to measure the difference between each spectrogram, because each point in the spectrogram means the amplitude of the corresponding frequency.

Chapter 4

Variational autoencoder based method

4.1 Preliminaries: Deterministic autoencoders and latent variables

The second method we propose is based on the variational autoencoder (VAE). VAE is a method based on deep learning and Bayesian inference. Since it is a method based on deep learning, hyperparameters tuning is always a necessary step. In this paper, we focus on the mathematical mechanism of VAE. The tricks used for tuning are not the key point.

Before knowing what's VAE, some basic knowledge about autoencoder is needed. An autoencoder is a type of neural network with an architecture shown in figure 4.1.

It contains two parts, an encoder and a decoder. The encoder learns the representation $\mathbf{z}$ of the data $\mathbf{x}$ in reduced dimensions. We call this new representation $\mathbf{z}$ latent variable, namely the variable that cannot be directly observed. In autoencoders, the latent variable is deterministic. And the decoder learns to reconstruct the input based on the latent variable $\mathbf{z}$, which is its output $\tilde{\mathbf{x}}$. For the decoder, this is a regression task, so usually the loss function is Mean-Square-Error (MSE) loss, $\mathcal{L} = \left\|\mathbf{x} - \tilde{\mathbf{x}}\right\|^2$. Mathematical formulations of the mechanism are equations (4.1), (4.2), (4.3).

$$\mathbf{z} = f_\phi(\mathbf{x}) \tag{4.1}$$

$$\tilde{\mathbf{x}} = f_\psi(\mathbf{z}) \tag{4.2}$$

$$\mathcal{L} = \|\mathbf{x} - \tilde{\mathbf{x}}\|^2 \tag{4.3}$$

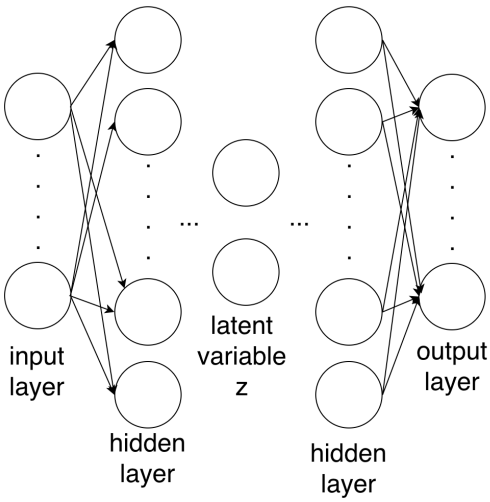


Figure 4.1: One example of autoencoder architecture

where f_ϕ and f_ψ represent the encoder and the decoder, respectively.

The ϕ and ψ are the parameters of the encoder and the decoder. They are supposed to be learned during the training process.

After the training process, given new data $\mathbf{x}'$, the encoder gives the corresponding representation $\mathbf{z}'$. The decoder generates reconstructed input $\hat{\mathbf{x}}$. The learned representation $\mathbf{z}'$ can be used for classification or regression if the autoencoder is well-trained.

Autoencoder is an unsupervised learning method. The purpose of using autoencoder is to reduce the dimensionality of the original input data, which is similar to the purpose of using Principle Component Analysis (PCA). The only difference is that PCA reduces the dimension using linear projection and autoencoder achieve the same goal in a non-linear way because of the non-linear activation $\sigma(x)$ in each neuron. This is an advantage when the data is complicated or high dimensional. And if we change the non-linear activation function to a linear one, *e.g.* identity function $f(x) = x$, then after training the latent variable $\mathbf{z}$ is the same as the output of PCA given new data.

4.2 Derivation of loss function in variational autoencoders

From a mathematical point of view, VAE has little to do with the autoencoder, although they have a similar architecture, which is the bottleneck-shape in the middle hidden layer. VAE is based on a probabilistic model. The latent variables are no more deterministic. The learned

latent variables from VAE represent probability distributions, usually assumed to be Gaussian distribution. This enables VAE to generate new data that doesn't exist in the dataset by sampling from the latent variable distribution. So VAE is a powerful model.

Before the detailed explanations of the VAE's mechanism, some definitions and settings are needed. Let us consider the data set $X = \{\mathbf{x}^1, \mathbf{x}^2, \cdots \mathbf{x}^N\}$ consisting of N samples from variable $\mathbf{x}$. We assume that the $\mathbf{x}$ is generated from a latent variable $\mathbf{z}$ and $\mathbf{z}$ is generated from some prior distribution $p_{\theta^\star}(\mathbf{z})$. We don't know the true parameter $\theta^\star$ and the true latent variable $\mathbf{z}$, so we assume that the prior $p_{\theta^\star}(\mathbf{z})$ and $p_{\theta^\star}(\mathbf{x}|\mathbf{z})$ come from parametric families of distributions $p_\theta(\mathbf{z})$ and $p_\theta(\mathbf{x}|\mathbf{z})$ [14]. The goal of VAE is to maximize the marginal likelihood $p_\theta(\mathbf{x})$, expressed by equation (4.4).

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z}) d\mathbf{z} \tag{4.4}$$

To calculate this integral, one possible method is Monte Carlo. The integral is approximated by sampling as many as possible values from the prior distribution $p(\mathbf{z})$ and calculate corresponding $p_\theta(\mathbf{x}|\mathbf{z})$, as equation (4.5) shows. The more we sample, the more accurate approximation we have.

$$p_\theta(\mathbf{x}) \approx \frac{1}{N} \sum_{i=1}^{N} p_\theta(\mathbf{x}|\mathbf{z_i}) p_\theta(\mathbf{z_i}) \tag{4.5}$$

where N is the number of samples.

However, if we use this method, then it is time-consuming and inefficient because of the large search space of $\mathbf{z}$. $p(\mathbf{x})$ is intractable, so the posterior $p_\theta(\mathbf{z}|\mathbf{x})$ is also intractable. We introduce $q_\phi(\mathbf{z}|\mathbf{x})$ to approximate the true posterior $p_\theta(\mathbf{z}|\mathbf{x})$.

$q_\phi(\mathbf{z}|\mathbf{x})$ is the probability distribution of $\mathbf{z}$ given the data $\mathbf{x}$, which is interpreted as an probabilistic encoder in VAE. The parameters in this encoder are represented by ϕ. And $p_\theta(\mathbf{x}|\mathbf{z})$ is the likelihood function, usually assumed to be distributed as Gaussian $\mathcal{N}(x|\hat{x}(\mathbf{z}, \theta), \sigma^2)$. In VAE it severs as a probabilistic decoder that outputs $\hat{x}$. The parameters in this decoder are represented by θ.

To measure the performance of the approximation of the posterior, we introduce the KL-divergence between the approximation of the posterior $q_\phi(\mathbf{z}|\mathbf{x})$ and the true posterior $p_\theta(\mathbf{z}|\mathbf{x})$, as shown in equation (4.6).

$$\begin{aligned} D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z}|\mathbf{x})) &= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})} d\mathbf{z} \\ &= \int q_\phi(\mathbf{z}|\mathbf{x}) \log q_\phi(\mathbf{z}|\mathbf{x}) d\mathbf{z} - \int q_\phi(\mathbf{z}|\mathbf{x}) \log p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z} + \log p(\mathbf{x}) \end{aligned} \tag{4.6}$$

Thus, we have

$$\log p(\mathbf{x}) - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z}|\mathbf{x})) = \mathcal{L}(\theta, \phi, \mathbf{x}) \tag{4.7}$$

where $\mathcal{L}(\theta, \phi, \mathbf{x}) = \int q_\phi(\mathbf{z}|\mathbf{x}) \log p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z} - \int q_\phi(\mathbf{z}|\mathbf{x}) \log q_\phi(\mathbf{z}|\mathbf{x}) d\mathbf{z}$.

Since $D_{KL}(p(x) \,||\, q(x)) \geq 0$, $\mathcal{L}(\theta, \phi, \mathbf{x})$ is smaller or equal to $\log p(\mathbf{x})$. It is the lower bound of $\log p(\mathbf{x})$. We call it evidence lower bound ($ELBO$). If we optimize the right-hand side of $ELBO$ w.r.t θ and ϕ using stochastic gradient descent (SGD) to find the maximum of $ELBO$, then the left-hand side of equation (4.7) also achieves its maximum. $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z}|\mathbf{x}))$ is minimum. Assuming that $q_\phi(\mathbf{z}|\mathbf{x})$ is a high capacity model that hopefully matches $p_\theta(\mathbf{z}|\mathbf{x})$, then the minimum of $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z}|\mathbf{x}))$ is 0 [14]. Instead of using intractable $p_\theta(\mathbf{z}|\mathbf{x}))$, we have a tractable approximation $q_\phi(\mathbf{z}|\mathbf{x})$.

After a few steps, $\mathcal{L}(\theta, \phi, \mathbf{x})$ is rewritten as equation (4.8). It is also the loss function in VAE.

$$
\begin{aligned}
\mathcal{L}(\theta, \phi, \mathbf{x}) &= \int q_\phi(\mathbf{z}|\mathbf{x}) \log[p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z})] d\mathbf{z} - \int q_\phi(\mathbf{z}|\mathbf{x}) \log q_\phi(\mathbf{z}|\mathbf{x}) d\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \log[p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z})] d\mathbf{z} - \int q_\phi(\mathbf{z}|\mathbf{x}) \log q_\phi(\mathbf{z}|\mathbf{x}) d\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \log p_\theta(\mathbf{x}|\mathbf{z}) d\mathbf{z} + \int q_\phi(\mathbf{z}|\mathbf{x}) \log p_\theta(\mathbf{z}) d\mathbf{z} - \int q_\phi(\mathbf{z}|\mathbf{x}) \log q_\phi(\mathbf{z}|\mathbf{x}) d\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \log p_\theta(\mathbf{x}|\mathbf{z}) d\mathbf{z} + \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{p_\theta(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} d\mathbf{z} \\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z}))
\end{aligned} \tag{4.8}
$$

4.2.1 Closed-form KL divergence

Based on equation (4.8), we know that the $ELBO$ has two components. One is the KL divergence of the posterior approximation from the prior. Another component is the expectation of the likelihood. The KL term can often be integrated analytically.

In VAE, we assume that the prior distribution $p(\mathbf{z})$ is a standard normal distribution $\mathcal{N}(0, I)$ and the posterior distribution $q_\phi(\mathbf{z}|\mathbf{x})$ is another Gaussian distribution $\mathcal{N}(\mu_\phi(\mathbf{x}), \Sigma_\phi(\mathbf{x}))$. The mean $\mu_\phi(\mathbf{x})$ and the covariance matrix $\Sigma_\phi(\mathbf{x})$ are the output of the encoder of VAE. The covariance matrix $\Sigma_\phi(\mathbf{x})$ is assumed to be a k dimensional diagonal matrix.

In this case, the KL divergence term has a closed form solution [6], as shown in equation (4.10).

$$
\begin{aligned}
D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p(\mathbf{z})) &= D_{KL}(\mathcal{N}(\mu_\phi(\mathbf{x}), \Sigma_\phi(\mathbf{x})) \,||\, \mathcal{N}(0, I)) && (4.9) \\
&= \frac{1}{2}(\mathrm{Tr}(\Sigma_\phi(\mathbf{x})) + \mu_\phi(\mathbf{x})^T \mu_\phi(\mathbf{x}) - k - \log\det(\Sigma_\phi(\mathbf{x}))) && (4.10)
\end{aligned}
$$

where k is the dimensionality of latent variable $\mathbf{z}$.

4.2.2 Reformulation of the reconstruction loss

Since we assume that the likelihood function $p_\theta(\mathbf{x}|\mathbf{z})$ is distributed as Gaussian $\mathcal{N}(\mathbf{x}|\hat{x}(\mathbf{z},\theta),\sigma^2)$, where σ is fixed for simplicity. So we have

$$\log p_\theta(\mathbf{x}|\mathbf{z}) = -\log\sigma - \frac{m}{2}\log(2\pi) - \frac{\|\mathbf{x}-\hat{\mathbf{x}}(\mathbf{z},\theta)\|^2}{2\sigma^2} \tag{4.11}$$

where m is the input dimension.

Based on equation (4.8) and (4.11), the ELBO is reformulated in equation (4.12).

$$\begin{aligned}\mathcal{L} &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[-\log\sigma - \frac{m}{2}\log 2\pi - \frac{\|\mathbf{x}-\hat{\mathbf{x}}(\mathbf{z},\theta)\|^2}{2\sigma^2}] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z})) \\ &= -\frac{1}{2\sigma^2}\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\|\mathbf{x}-\hat{\mathbf{x}}(\mathbf{z},\theta)\|^2] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z})) + C\end{aligned} \tag{4.12}$$

where C is constant.

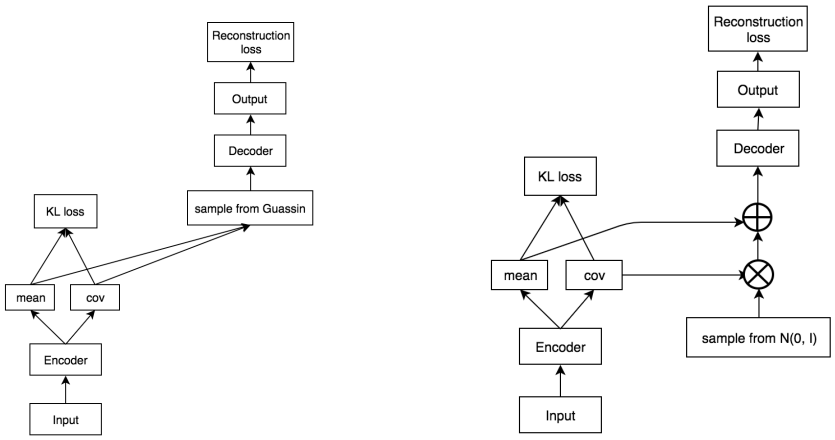
We have constant C in equation (4.12), because $-\log\sigma$ and $-\frac{m}{2}\log 2\pi$ are constants and can be moved out of the expectation. Thus, we have another interpretation for the VAE loss $\mathcal{L}$. As shown in equation (4.12), the mean-square-error measures the performance of the encoder and the decoder in VAE. The KL divergence term is for the regularization of ϕ, which in our case is to encourage the approximation $q_\phi(\mathbf{z}|\mathbf{x})$ to be similar to the prior $p_\theta(\mathbf{z})$.

4.3 Reparameterization trick in VAE

As shown in figure 4.2a, $\mathbf{z}$ is sampled from Gaussian distribution $\mathcal{N}(\mu_\phi(\mathbf{x}),\Sigma_\phi(\mathbf{x}))$ and it is the input for the decoder of VAE. The sampling operation is not differentiable. However, the training of the VAE is based on the backward propagation and gradient descent. The parameter ϕ in the encoder cannot get updated. To avoid this situation, the reparameterization trick is used.

The reparameterization trick is to represent a random variable z using samples ϵ from a base distribution $p(\epsilon)$ with differentiable transformations, denoted by $g(\cdot)$. Equation (4.12) shows that to maximize the $\mathcal{L}$, the calculation of expectation $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\|\mathbf{x}-\hat{\mathbf{x}}\left(\mathbf{z},\theta\right)\|^2\right]$ is required, which means that sampling from $q_\phi(\mathbf{z}|\mathbf{x})$ is necessary.

To represent $\mathbf{z} \sim \mathcal{N}\left(\mu_\phi\left(\mathbf{x}\right),\Sigma_\phi\left(\mathbf{x}\right)\right)$ using $\epsilon \sim \mathcal{N}(0,\mathbf{I})$, where $\Sigma_\phi(\mathbf{x})$ is the diagonal covariance matrix, we define our differentiable transformation $g(\epsilon) = \mu_\phi(\mathbf{x}) + \Sigma_\phi^{\frac{1}{2}}(\mathbf{x})\epsilon$. Based on this



(a) VAE without Reparameterization (b) VAE with Reparameterization

Figure 4.2: VAE with and without reparameterization trick

definition, we have $\mathbf{z} = g(\epsilon)$. We denote $\|\mathbf{x} - \hat{\mathbf{x}}(\mathbf{z}, \theta)\|^2$ as $f(\mathbf{z})$. Thus, we have

$$\begin{aligned}
\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\|\mathbf{x} - \hat{\mathbf{x}}(\mathbf{z}, \theta)\|^2\right] &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[f(\mathbf{z})\right] \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) f(\mathbf{z})\, d\mathbf{z} \\
&= \int \det(2\pi\Sigma)^{-\frac{1}{2}} \mathrm{e}^{-\frac{1}{2}(\mathbf{z}-\mu)^T\Sigma^{-1}(\mathbf{z}-\mu)} f(\mathbf{z})\, d\mathbf{z} \\
&= \int \det(2\pi\Sigma)^{-\frac{1}{2}} \mathrm{e}^{-\frac{1}{2}\epsilon^T\epsilon} f\left(g\left(\epsilon\right)\right) \det(\frac{\partial g}{\partial \epsilon})\, d\epsilon \\
&= \int \det(2\pi\Sigma)^{-\frac{1}{2}} \mathrm{e}^{-\frac{1}{2}\epsilon^T\epsilon} f\left(g\left(\epsilon\right)\right) \det(\Sigma^{\frac{1}{2}})\, d\epsilon \\
&= \int (2\pi)^{-\frac{k}{2}} \mathrm{e}^{-\frac{1}{2}\epsilon^T\epsilon} f\left(g\left(\epsilon\right)\right)\, d\epsilon \\
&= \int p(\epsilon) f\left(g\left(\epsilon\right)\right)\, d\epsilon \\
&= \mathbb{E}_{p(\epsilon)}\left[f\left(g\left(\epsilon\right)\right)\right] \\
&= \mathbb{E}_{p(\epsilon)}\left[\left\|\mathbf{x} - \hat{\mathbf{x}}\left(\mu_\phi(\mathbf{x}) + \Sigma_\phi^{\frac{1}{2}}\epsilon, \theta\right)\right\|^2\right] \qquad (4.13)
\end{aligned}$$

where k is the dimensionality of the latent variable.

By using the chain rule based on equation 4.13, we are able to calculate the gradient w.r.t θ and ϕ, because $g(\cdot)$ is differentiable and the randomness introduced by sampling now exists in

ϵ and doesn't influence gradient calculation. Figure 4.2 illustrates the VAE without reparameterization and the VAE with reparameterization. In summary, by using the reparameterization trick, the undifferentiable operation sampling is avoided. The gradient flows from the output layer of VAE to the input layer of VAE.

Chapter 5

Squared-loss mutual information based method

5.1 Mutual information and problem definition

Assume $\mathbf{x} \in \mathbb{R}^d$ is one input data, $y \in \{+1, -1\}$ is the corresponding label and $p(\mathbf{x}, y)$ is joint density. In machine learning task, we are usually given data $\mathbf{x}$ and want to predict corresponding label y. In other words, we want to measure how much $\mathbf{x}$ tells us about y. One quantity used to measure this is mutual information (MI). It is defined by equation (5.1).

$$MI := \sum_{y=\pm 1} \int p(\mathbf{x}, y) \log \frac{p(\mathbf{x}, y)}{p(\mathbf{x})p(y)} d\mathbf{x} \tag{5.1}$$

If MI is high, then given $\mathbf{x}$, the uncertainty of y is largely reduced. If MI is low, then given $\mathbf{x}$, it is a small reduction of uncertainty of y. If MI is 0, then the random variable $\mathbf{x}$ and y are independent, which means that knowing $\mathbf{x}$ doesn't help us to know y.

Comparing with the definition of Kullback-Leibler divergence, shown in equation (5.2). It is clear that MI is the Kullback-Leibler divergence from the joint density $p(\mathbf{x}, y)$ to the product of $p(\mathbf{x})$ and $p(y)$.

$$D_{KL} = D_{KL}(P||Q) = \int P \log \frac{P}{Q}, \tag{5.2}$$

Kullback-Leibler divergence measures the dissimilarity between distribution P and distribution Q. Based on Kullback-Leibler divergence, we have another perspective to understand MI. MI measures how different it is between the joint density $P(\mathbf{x}, y)$ and the product of $P(\mathbf{x})$ and $P(y)$.

But empirically approximating MI from continuous data is not straightforward [25]. Squared-loss MI (SMI) is proposed to solve this problem.

$$SMI := \sum_{y \pm 1} \frac{p(y)}{2} \int (\frac{p(\mathbf{x}, y)}{p(\mathbf{x})p(y)} - 1)^2 p(\mathbf{x}) d\mathbf{x} \tag{5.3}$$

SMI is the Pearson divergence from the joint density $p(\mathbf{x}, y)$ to the product of the marginals $p(\mathbf{x})p(y)$. In paper [25], the authors mentioned that the Pearson divergence and the Kullback-Leibler divergence belong to the class of f-divergences, which means that they share similar properties. In summary, we use SMI to approximate MI in practical uses for simplicity. By this approximation, we evaluate the dependency between $\mathbf{x}$ and y.

As mentioned in chapter 2, the label we have shows the quality of the final product, steering gear. The defect final product can be caused by the defect BNA, other defect components or both. The relationship is shown in figure 5.1. From the probabilistic perspective, the defect

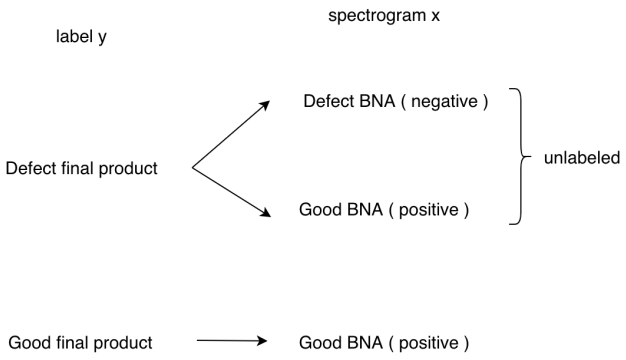


Figure 5.1: Relationship between BNA and final product

final product is a mixture of two probability distributions. One is a distribution of good BNA and another distribution is defect BNA. Points that are sampled from this mixture have the same label, representing the defect. In this scenario, the data is called Positive-Unlabel (PU) data.

PU data is mathematically defined as:

$$\{\mathbf{x}_i^P\}_{i=1}^{n_P} \sim p(\mathbf{x}|y = +1)$$
$$\{\mathbf{x}_i^U\}_{i=1}^{n_U} \sim p(\mathbf{x}) = \theta_P p(\mathbf{x}|y = +1) + \theta_N p(\mathbf{x}|y = -1)$$

where $\theta_P := p(y = +1)$ and $\theta_N := p(y = -1)$ are the class-prior probabilities.

The positive data has a label of $+1$. The unlabel data is a mixture of positive data with label $+1$ and negative data with label -1.

In ThyssenKrupp data situation, the defect final product corresponds to the unlabel data, because it is a mixture of the good BNA (positive data) and the defect BNA (negative data). The good final product corresponds to the positive data.

5.2 SMI derivation and approximation

Assuming we have PU data, then based on the law of total probability and the assumption $\theta_P + \theta_U = 1$, we have equation (5.4).

$$\begin{aligned}
\frac{\theta_N p(\mathbf{x}|y=-1)}{\theta_P p(\mathbf{x}|y=+1) + \theta_N p(\mathbf{x}|y=-1)} &= 1 - \frac{\theta_P p(\mathbf{x}|y=+1)}{\theta_P p(\mathbf{x}|y=+1) + \theta_N p(\mathbf{x}|y=-1)} \\
\theta_N(\frac{p(\mathbf{x}|y=-1)}{p(\mathbf{x})} - 1) &= \theta_P(1 - \frac{p(\mathbf{x}|y=+1)}{p(\mathbf{x})}) \\
(\frac{p(\mathbf{x}|y=-1)}{p(\mathbf{x})} - 1)^2 &= \frac{\theta_P^2}{\theta_N^2}(\frac{p(\mathbf{x}|y=+1)}{p(\mathbf{x})} - 1)^2
\end{aligned} \tag{5.4}$$

Thus we have

$$\begin{aligned}
SMI =& \frac{\theta_P}{2}\int(\frac{\theta_P p(\mathbf{x}|y=+1)}{p(\mathbf{x})} - 1)^2 p(\mathbf{x})d\mathbf{x} \\
&+ \frac{\theta_N}{2}\int(\frac{\theta_N p(\mathbf{x}|y=-1)}{p(\mathbf{x})} - 1)^2 p(\mathbf{x})d\mathbf{x}
\end{aligned} \tag{5.5}$$

Based on equation (5.4), we reformulate equation (5.5) and obtain equation (5.6)

$$\begin{aligned}
SMI &= \frac{\theta_P}{2}\int(\frac{\theta_P p(\mathbf{x}|y=+1)}{p(\mathbf{x})} - 1)^2 p(\mathbf{x})d\mathbf{x} + \frac{\theta_N}{2}\int\frac{\theta_P^2}{\theta_N^2}(\frac{p(\mathbf{x}|y=+1)}{p(\mathbf{x})} - 1)^2 p(\mathbf{x})d\mathbf{x} \\
&= \frac{\theta_P}{2\theta_N}\int(\frac{p(\mathbf{x}|y=+1)}{p(\mathbf{x})} - 1)^2 p(\mathbf{x})d\mathbf{x}
\end{aligned} \tag{5.6}$$

According to paper [23], there exists a lower bound of SMI for any function $w(\mathbf{x})$

$$SMI \geq \frac{\theta_P}{\theta_N}(-J(w) - \frac{1}{2}), \tag{5.7}$$

where

$$J(w) := \frac{1}{2}\int w^2(\mathbf{x})p(\mathbf{x})d\mathbf{x} - \int w(\mathbf{x})p(\mathbf{x}|y=+1)d\mathbf{x}$$

the equality holds if and only if

$$w(\mathbf{x}) = \frac{p(\mathbf{x}|y=+1)}{p(\mathbf{x})} \tag{5.8}$$

Function $w(\mathbf{x})$ can be a complicated machine learning model, for example, a neural network, or a simple model, a linear regression model. The approximation of $J(w)$ is equation (5.9), according to paper [23].

$$\hat{J}(w) := \frac{1}{2n_U}\sum_{k=1}^{n_U} w^2(\mathbf{x}_k^U) - \frac{1}{n_P}\sum_{i=1}^{n_P} w(\mathbf{x}_i^P) \tag{5.9}$$

Correspondingly, we have an approximation of SMI, namely $\widehat{SMI}$, as equation (5.10) shows. The $\frac{\theta_P}{\theta_N}$ is a proportional constant. The maximum of $\widehat{SMI}$ only depends on $\hat{J}$, not on the proportion constant. This is an advantage, since we don't need to have any assumption over the mixing coefficients.

$$\widehat{SMI} = \frac{\theta_P}{\theta_N}(-\hat{J}(\hat{w}) - \frac{1}{2}) \tag{5.10}$$

5.3 Representation learning and classification using SMI

Dimension reduction is defined by a mapping $v : \mathbb{R}^d \to \mathbb{R}^m$, where $m < d$. And if the mapping v satisfies the condition

$$p(y|\mathbf{x}) = p(y|v(\mathbf{x})) \tag{5.11}$$

then the obtained result $v(\mathbf{x})$ can be used instead of the original high-dimensional data $\mathbf{x}$ [23]. The function v that satisfies the condition (5.11) is called sufficient dimension reduction. In paper [23], the author claims that maximizing the SMI is finding sufficient representation for the output y. So the goal is to maximize the SMI with respect to the parameters in mapping v and function w and preserve the dependency between input $\mathbf{x}$ and output y. We consider $v(\mathbf{x})$ and $w(\mathbf{x})$ as different layers in a neural network, then the loss function L of this neural network is equation (5.12).

$$L = \frac{1}{2n_U}\sum_{k=1}^{n_U} w^2(v(\mathbf{x}_k^U)) - \frac{1}{n_P}\sum_{i=1}^{n_P} w(v(\mathbf{x}_i^P)) \tag{5.12}$$

SMI not only gives us representation but also provides the classification based on the learned representation. Equation (5.8) gives us insights about how the output of our neural network $w(\mathbf{x})$ relates with the classification prediction $p(y=+1|\mathbf{x})$ made by the neural network and the unlabel data distribution $p(\mathbf{x})$.

By using Bayes' theorem, we have

$$p(\mathbf{x}|y=+1) = \frac{p(y=+1|\mathbf{x})p(\mathbf{x})}{p(y=+1)} = \frac{p(y=+1|\mathbf{x})p(\mathbf{x})}{\theta_P} \tag{5.13}$$

so we have another equation (5.14) based on equation (5.8) and (5.13).

$$w(\mathbf{x}) = \frac{p(y=+1|\mathbf{x})}{\theta_P} \tag{5.14}$$

Now we know that the output $w(\mathbf{x})$ of the neural network is the classification prediction $p(y=+1|\mathbf{x})$ divided by θ_P, which is the mixing coefficient for positive data in the unlabel data. Based on this, a general overview of our neural network is shown in figure 5.2. The learned representation $v(\mathbf{x})$ is one hidden layer in the neural network. The activation function of these neurons is *Relu*. The prediction $p(y=+1|\mathbf{x})$ is made by one neuron with Sigmoid activation. Then it is only multiplied by weight ϕ and without adding bias b. The result is given to the output layer, which is one neuron with a linear activation function. The output $w(\mathbf{x})$ of this neural network is $\phi * p(y=+1|\mathbf{x})$. If the neural network is well trained, then the value of ϕ approximates the inverse of θ_P, according to equation (5.14).

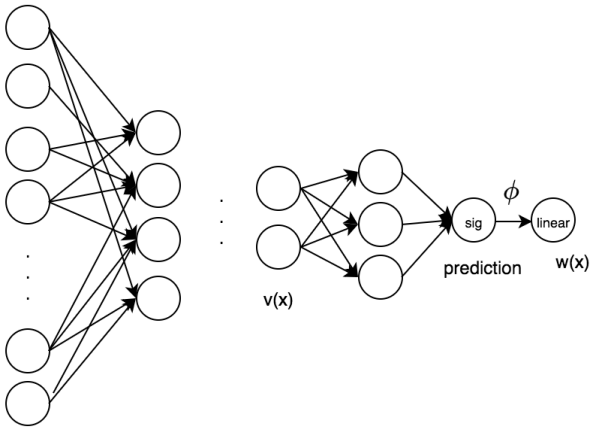


Figure 5.2: An overview of the neural network for SMI method

Chapter 6

Experiments and evaluation

6.1 Experiments using standard datasets

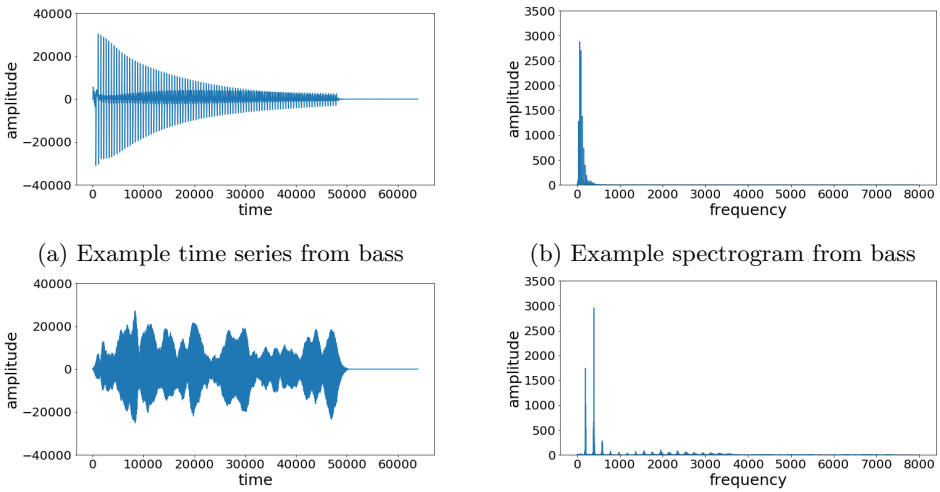
We use NSynth dataset in our experiments for time-delayed embedding method. For VAE method and SMI method, we use MNIST dataset. On the one hand, we use these datasets to check whether our implementations are correct or not. On the other hand, the label of BNA is not available. It is necessary to use the standard datasets to see our methods' capabilities.

NSynth dataset is from [7]. The dataset contains musical notes generated from different sources, including bass, string, and other instruments. For simplicity, we only choose musical notes generated by bass and human. And to reduce the computation cost, we randomly sample 288 audios from all the audio files and form a new data set. There are 147 audios generated by a bass. The rest 141 audios are vocal. Some examples and corresponding spectrogram after real Fourier transform are shown in figure 6.1. One remark is that each spectrogram is composed of 64000 points representing frequencies and corresponding amplitudes. And the frequency interval is between 0 Hz to 8000 Hz.

The MNIST dataset was firstly generated by Yann, Lecun [17]. Each image has 784 pixels and contains one hand-written digit from 0 to 9. It has already been widely used for training and testing of models in machine learning. Some example digits from MNIST dataset are shown in figure 6.2.

6.1.1 Time delayed embedding method

In this section, we illustrate the results based on time domain information using time delayed embedding and the results based on frequency domain information, which is the spectrogram.



(a) Example time series from bass

(b) Example spectrogram from bass

(c) Example time series from vocal

(d) Example spectrogram from vocal

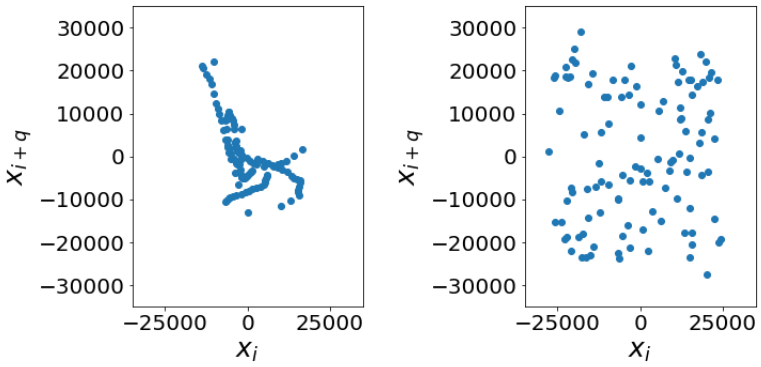
Figure 6.1: Example time series and spectrogram of bass and vocal audio

Figure 6.2: Example digits from MNIST

Based on time domain

Firstly, we choose $k = 2$ in equation (3.1) so that the time block is a set of 2-dimensional points. We visualize some of these time blocks to see whether the parameters are properly chosen. Figure 6.3 shows example time blocks of time series generated by a bass and by a human. The result already shows the difference between the bass time series and the vocal time series. The points in the time block generated from the bass time series are more clustered, while the points in the time block generated from vocal time series are more scattered.

The learned representation based on time delayed embedding with $k = 10, q = 1000$ is shown in figure 6.4. The learned representations of the bass and the vocal audios are clearly in a separable situation.



(a) Example time block of bass audio

(b) Example time block of vocal audio

Figure 6.3: Example time blocks generated from bass and vocal audio

Based on frequency domain

Since each spectrogram is composed of 64000 points, directly calculating the Wasserstein distance between measures is time-consuming and requires computation power. To reduce the computation cost, we sample every 500th point from the spectrogram, which is the input for our Wasserstein distance calculation. The examples of sampled spectrograms are shown in figure 6.5.

The learned representation based on Wasserstein distance and MDS is shown in figure 6.6. The learned representation clearly shows a cluster of black points, while the orange points are distributed widely.

6.1.2 VAE method

We choose two latent variables z_1 and z_2. These two latent variables form the latent space. Theoretically, the number of latent variables can be any value. Here we make it 2 for a better visualization. The encoder has two hidden layers. The first hidden layer has 512 neurons and the second hidden layer has 128 neurons. There are also two hidden layers in the decoder. The first hidden layer has 128 neurons and the second has 512 neurons. The activation function of neurons in hidden layers is *Relu*. The output layer of the VAE uses a linear function as activation function since the decoder generates reconstructed input, which is a regression task. And we choose the learning rate equal to 0.005. For the optimizer, we use Adam optimization.

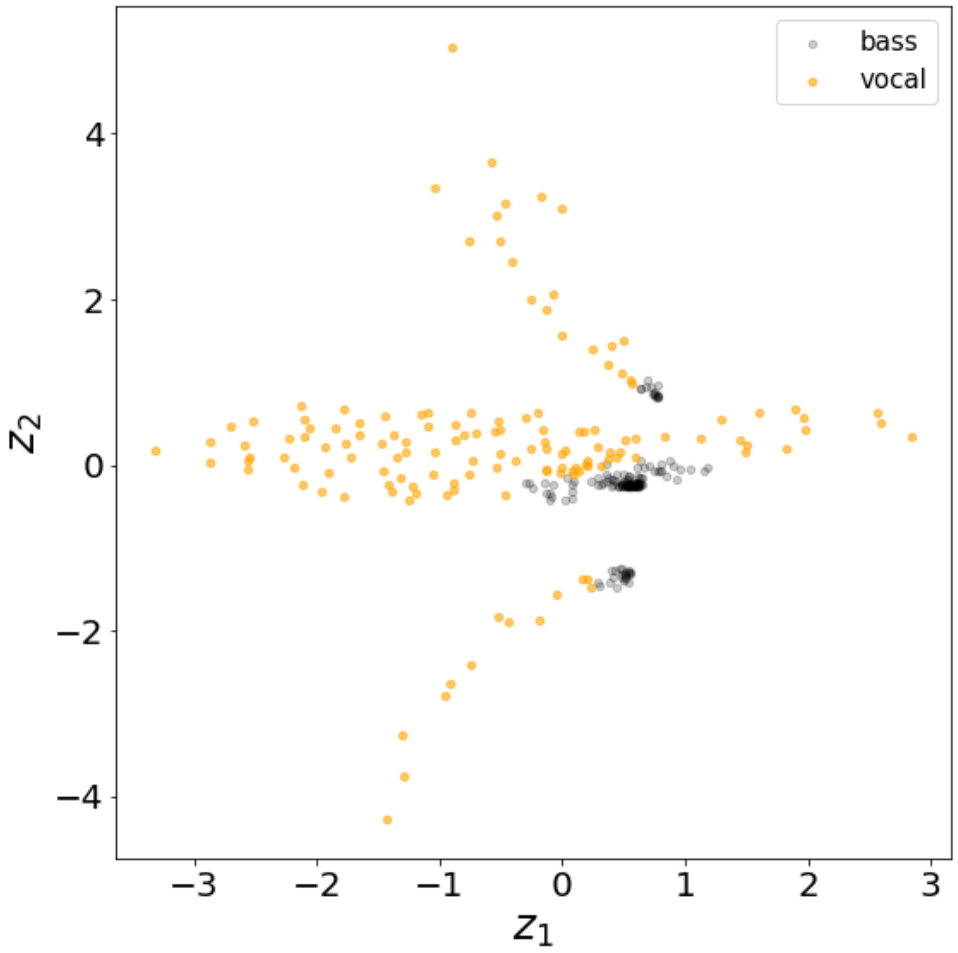
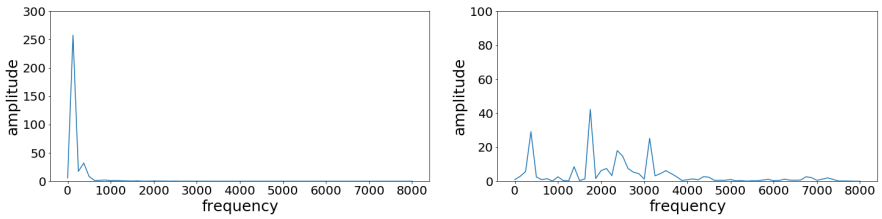


Figure 6.4: Learned representation based on time domain using bass and vocal audio



(a) Example input spectrogram of bass (b) Example input spectrogram of vocal

Figure 6.5: Example input from bass and vocal audio for Wasserstein distance calculation

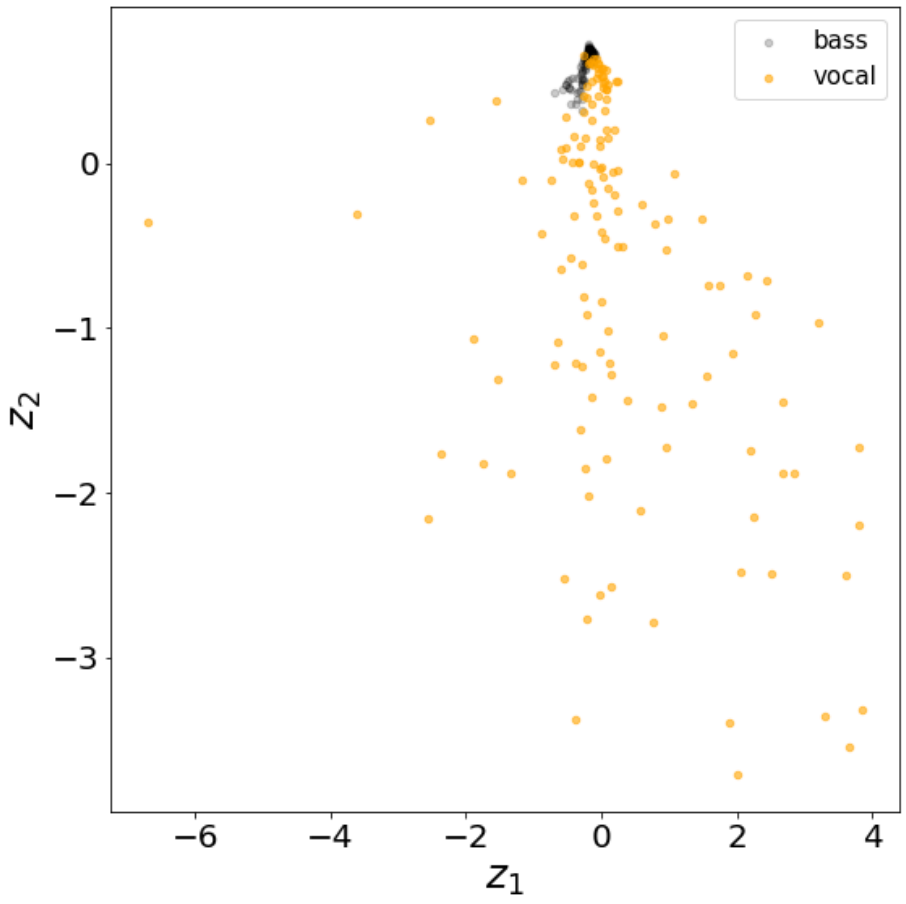


Figure 6.6: Learned representation based on frequency domain using bass and vocal audio

Data preprocessing

Each pixel value in the MNIST image is between 0 and 255. According to paper [18], convergence is faster if each input has an average close to 0. So each pixel value is divided by 255 in our

experiments.

The entire data is split into two sets. One is the training set for training the parameters in the neural network. Another is the test set for checking the performance of the neural network and tuning the hyperparameters in case of overfitting or underfitting situation.

Result visualization

The learned representation (z_1, z_2) of the test set is shown in figure 6.7. From the learned representation, based on the color bar on the right, it is clear to see the distributions of different digits.

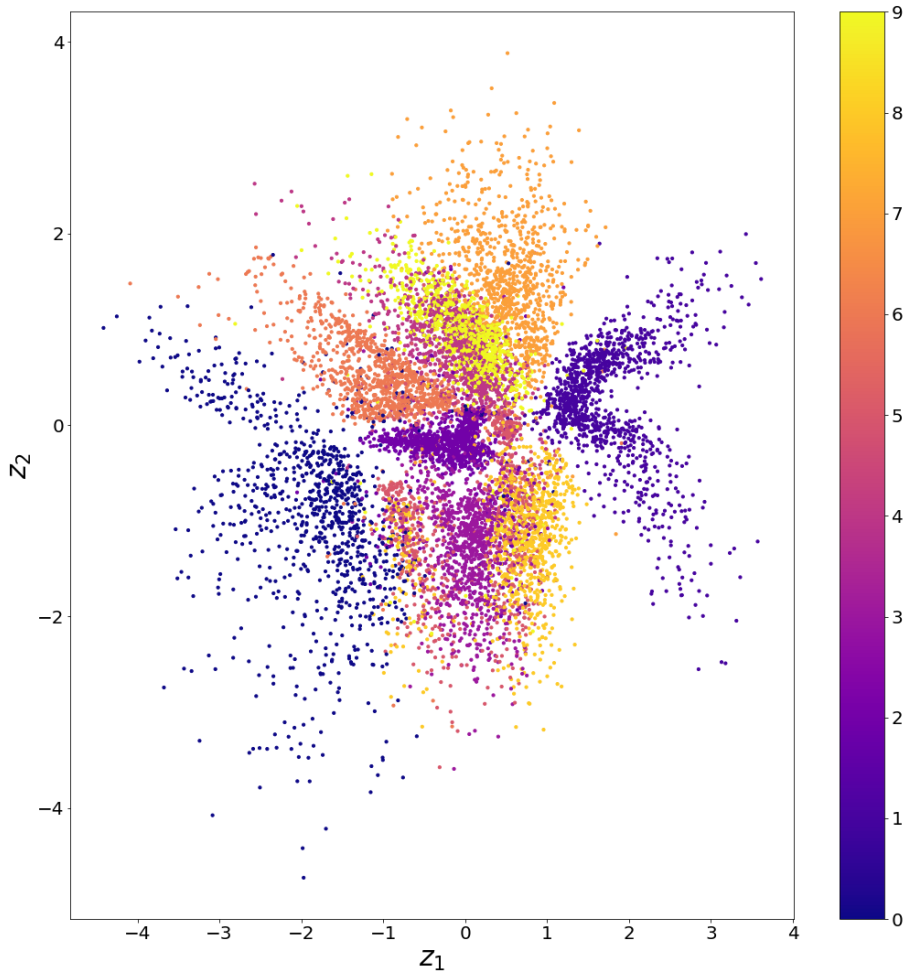


Figure 6.7: Learned representation of test set using MNIST dataset

Then we sample from the learned distribution $p(\mathbf{z}|\mathbf{x})$ and generate new images. Results are shown in figure 6.8. The first column shows the digits in the dataset. They are the input for the VAE. The title is the corresponding label. The rest columns show the generated digits. They are all generated by samples from the distribution $p(\mathbf{z}|\mathbf{x})$. The title of those images is the sampled value (z_1, z_2).

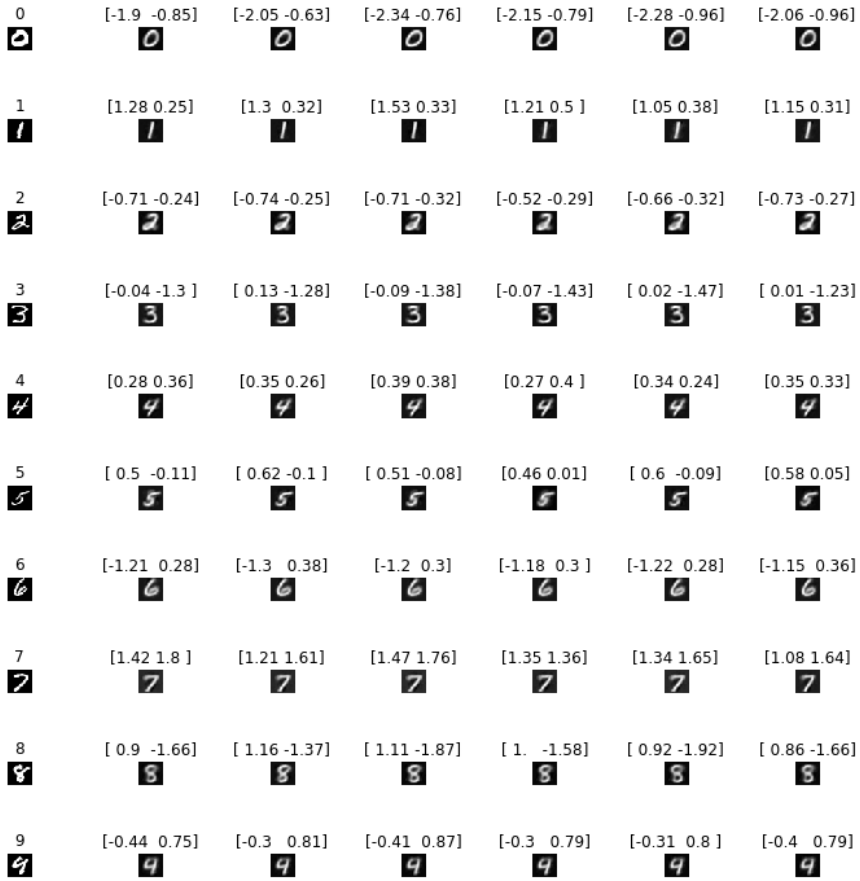


Figure 6.8: Generated digits using MNIST dataset

Based on these experiments using MNIST dataset, VAE gives us reasonable results. Firstly, it proves the VAE's capability of learning latent variables and generating new data using learned distribution. Secondly, we are sure that our implementation of VAE is correct.

6.1.3 SMI method

For checking our implementation of SMI method, we use MNIST dataset. Some preprocessing steps are done because SMI requires positive data and unlabel data, which is different from the data that VAE requires.

Firstly, based on the value of the digit, we reassign label to each digit. For the digit smaller or equal to 4, it is considered as positive data. For the digit larger than 4, it is considered as negative data. We generate unlabel data by sampling from positive data and negative data. The proportion of the positive data in the unlabel dataset is the mixing coefficient θ_P. By choosing different mixing coefficient θ_P of positive data in the unlabel data, we have different unlabel datasets. In our experiments $\theta_P \in \{0.3, 0.5, 0.7\}$, so we have three unlabel datasets.

In summary, the data we use for training and testing is positive data and unlabel data. To evaluate the performance of representation learning and classification, only unlabel data is used. The representation dimensionality is chosen to be 2. It is for a better visualization. Any dimensionality is theoretically workable here.

Experiments with $\theta_P = 0.3$

For $\theta_P = 0.3$, we design our neural network with 2 hidden layers, which have 60 neurons and 20 neurons, respectively. The dropout rate is 0.3. We also use $L2$ regularization of 0.01. After the layer that outputs the representation $v(\mathbf{x})$, there is a hidden layer with 10 neurons. Then it is the layer with Sigmoid activation function, which produces the prediction $p(y = +1|\mathbf{x})$. The prediction is given to a linear neuron and only multiplied with the weight ϕ. The output of the linear neuron is $w(\mathbf{x})$. The learning rate λ is 0.05 with a weight decay of 0.005. All the neurons in hidden layers have an activation function of *Relu*.

The learned representation (z_1, z_2) of unlabel data are plotted in a heatmap. The heatmap for unlabel data from test set is figure 6.9. The orange points are the learned representation of positive data in the unlabel dataset. The black points are the learned representation of negative data in the unlabel dataset. The deeper background color is, the higher value of $p(y = +1|\mathbf{x})$ is.

The inverse of the last layer's weight ϕ is 0.363. It is designed to approximate the mixing coefficient of positive data in the unlabel dataset. Our ground truth is $\theta_P = 0.3$, which means that 30 percent of unlabel data are digits smaller or equal to 4 and the rest are all larger than 4. So the approximation works. Other evaluations of performance are shown in table 6.1.

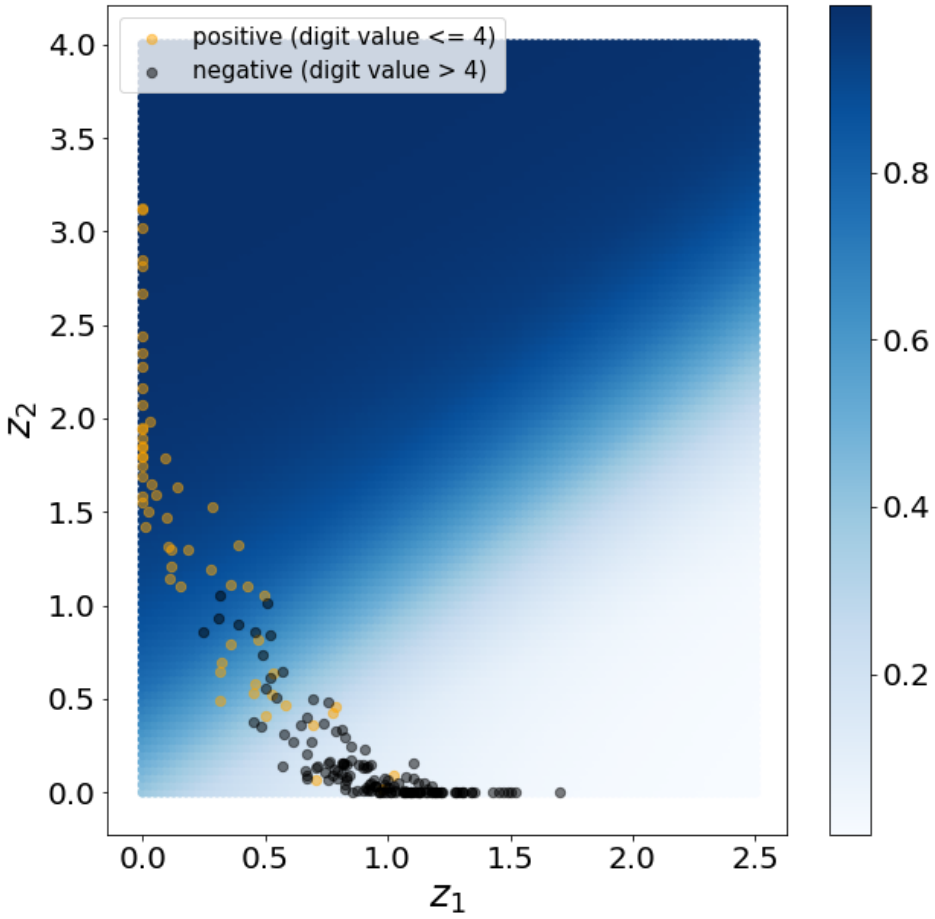


Figure 6.9: Learned representation of test set with $\theta_P = 0.3$ using MNIST dataset

Experiments with $\theta_P = 0.5$

Similar to experiments with $\theta_P = 0.3$, now the $\theta_P = 0.5$, which means that half data in the unlabel data are sampled from positive data and half are from negative data. The neural network remains the same architecture as the architecture in the experiment with $\theta_P = 0.3$. Other hyperparameters also remain the same.

The heatmap of unlabel data in the test set is shown in figure 6.10. The coloring scheme

Type	Num of data	Num of misclassification	Misclassification rate
Positive data $\mathbf{x}^P$	600	147	0.243
Negative data $\mathbf{x}^N$	1400	76	0.054
Unlabel data $\mathbf{x}$	2000	223	0.111

(a) Unlabel data in training set with $\theta_P = 0.3$

Type	Num of data	Num of misclassification	Misclassification rate
Positive data $\mathbf{x}^P$	60	11	0.183
Negative data $\mathbf{x}^N$	140	8	0.064
Unlabel data $\mathbf{x}$	200	19	0.095

(b) Unlabel data in test set with $\theta_P = 0.3$

Table 6.1: Number and rate of misclassification with $\theta_P = 0.3$ using MNIST dataset

is the same. In this experiment, $\frac{1}{\phi} = 0.413$. It still approximates the ground truth, $\theta_P = 0.5$. The number of misclassifications and misclassification rate are given in table 6.2.

Type	Num of data	Num of misclassification	Misclassification rate
Positive data $\mathbf{x}^P$	1000	327	0.327
Negative data $\mathbf{x}^N$	1000	60	0.060
Unlabel data $\mathbf{x}$	2000	387	0.194

(a) Unlabel data in training set with $\theta_P = 0.5$

Type	Num of data	Num of misclassification	Misclassification rate
Positive data $\mathbf{x}^P$	100	29	0.290
Negative data $\mathbf{x}^N$	100	6	0.06
Unlabel data $\mathbf{x}$	200	35	0.175

(b) Unlabel data in test set with $\theta_P = 0.5$

Table 6.2: Number and rate of misclassification with $\theta_P = 0.5$ using MNIST dataset

Experiments with $\theta_P = 0.7$

In the third experiment, $\theta_P = 0.7$. The positive data has a proportion of 0.7 in the unlabel dataset. The difficulty of representation learning and classification increases, because the proportion of negative data is only 0.3 in the unlabel dataset, which means that the unlabel dataset is similar to the positive dataset. So we change the number of neurons in the first hidden layer to 500. Other hyperparameters remain the same. The heatmap of unlabel data in the test set

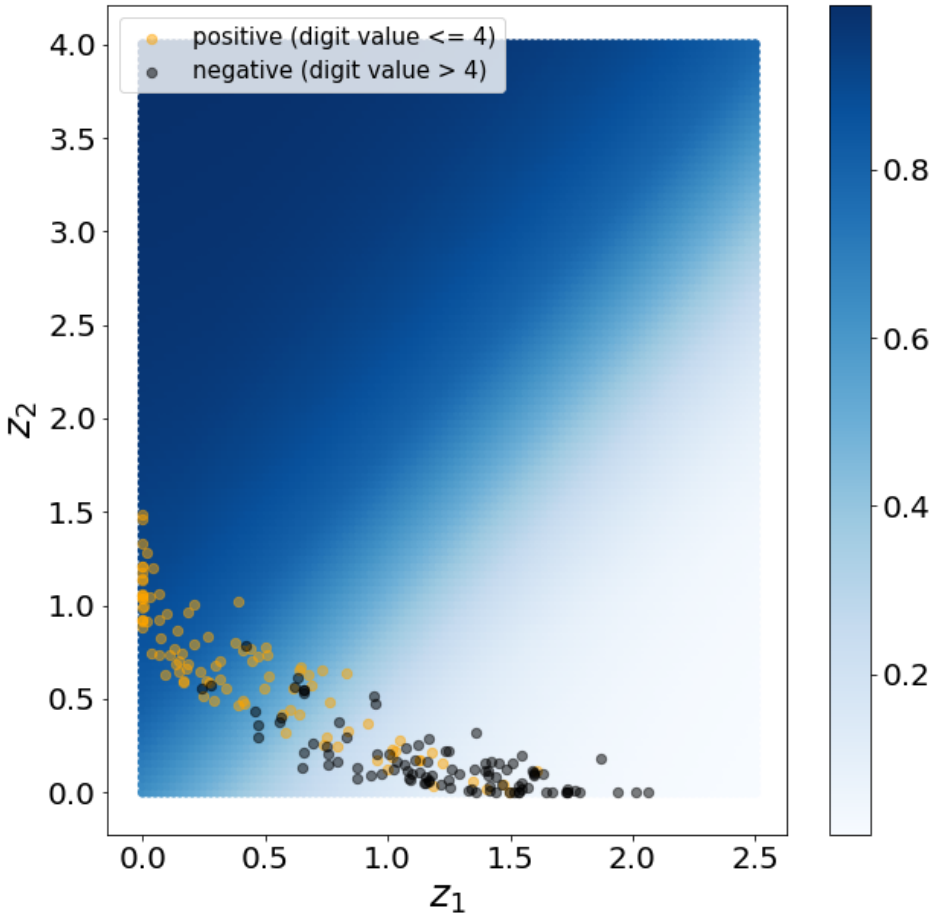


Figure 6.10: Learned representation of test set with $\theta_P = 0.5$ using MNIST dataset

is shown in figure 6.11. The number of misclassifications and misclassification rate are shown in table 6.3.

The approximation $\frac{1}{\phi}$ of θ_P now is 0.483, which varies from the ground truth 0.7. However, the classification result shows that the classifier has a low misclassification rate for negative data in the unlabel dataset, even though there are only 3% negative data in the training set.

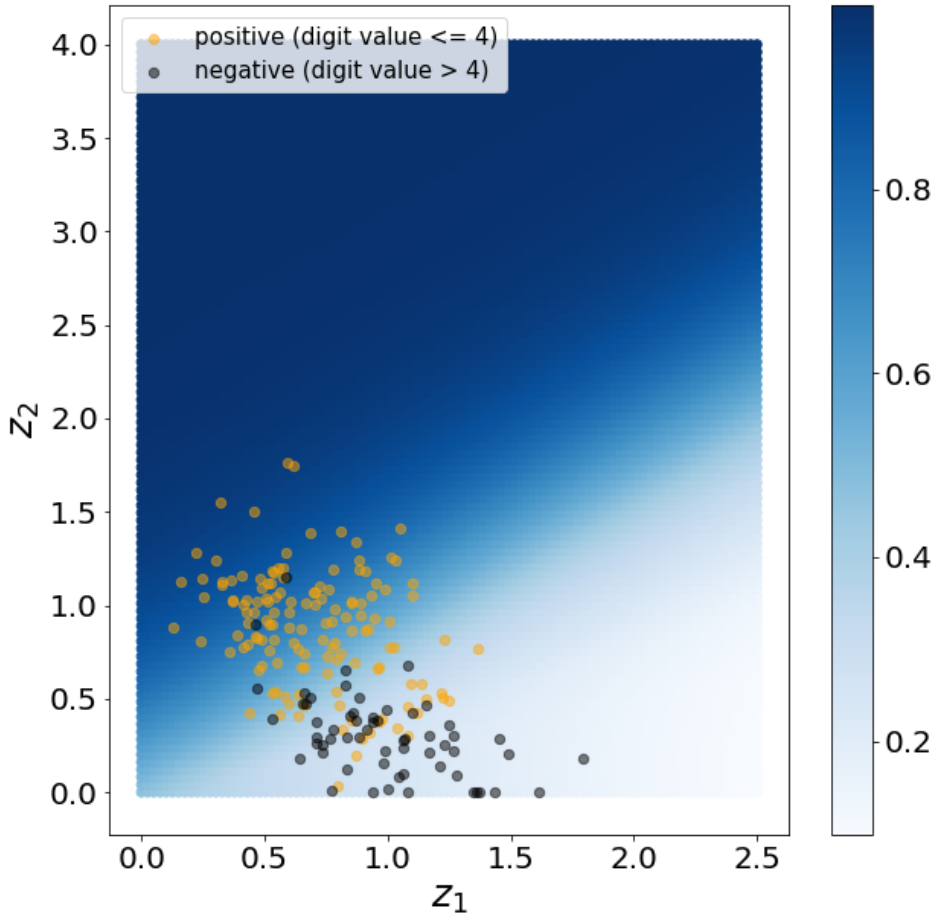


Figure 6.11: Learned representation of test set with $\theta_P = 0.7$ using MNIST dataset

6.2 Experiments using ThyssenKrupp dataset

Based on the experiments of Nsynth dataset and MNIST dataset, we are sure that the implementations of the proposed methods are all correct. In this section, we analyze the dataset from ThyssenKrupp.

Type	Num of data	Num of misclassification	Misclassification rate
Positive data $\mathbf{x}^P$	1400	536	0.383
Negative data $\mathbf{x}^N$	600	45	0.077
Unlabel data $\mathbf{x}$	2000	578	0.291

(a) Unlabel data in training set with $\theta_P = 0.7$

Type	Num of data	Num of misclassification	Misclassification rate
Positive data $\mathbf{x}^P$	140	32	0.229
Negative data $\mathbf{x}^N$	60	3	0.05
Unlabel data $\mathbf{x}$	200	20	0.175

(b) Unlabel data in test set with $\theta_P = 0.7$

Table 6.3: Number and rate of misclassification with $\theta_P = 0.7$ using MNIST dataset

6.2.1 General preprocessing of data

As discussed in chapter 2.1, each time series is generated by the rotation of one BNA. One defect BNA can be caused by only some parts of itself but not every part. So when the ball screw is rotated to some certain position, then the signal shows the features representing these parts are the cause. So we split each time series into 5 equal parts. The figure 6.12 is the original time series. The figure 6.13 shows the split 5 parts.

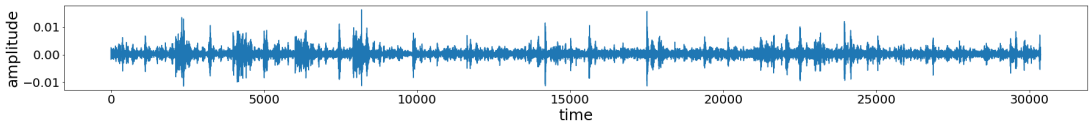
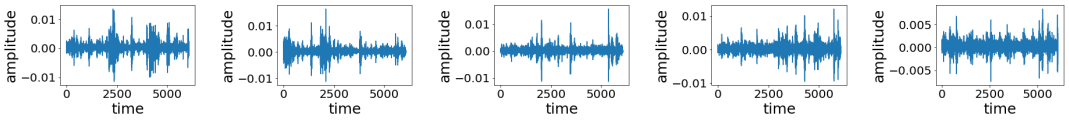


Figure 6.12: One original time series in ThyssenKrupp dataset

The number of original time series is 5616. And 172 of them are assembled into defect final product. The rest BNAs are assembled into the good final products. Each split time series have the same label as the original time series. So the proportion of the defect is $\frac{172}{5616} \approx 0.03$ for sub time series and original time series.

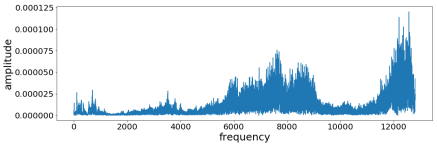
Another preprocessing we do is to use real Fourier transform on each sub time series. And correspondingly, we obtain spectrogram after this preprocessing. Each spectrogram is large and inefficient for further computation, so we do a 1-D convolution on the spectrogram using an averaging kernel with a size of 50 and a stride of 50. The dimensionality of the spectrogram is greatly reduced after the convolution. One example of the original spectrogram is shown in figure 6.14a. The corresponding convoluted result is shown in figure 6.14b.

After previous preprocessing steps, it is possible that one split time series or its spectrogram

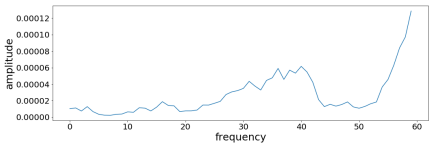


(a) First sub time series from ThyssenKrupp dataset (b) Second sub time series from ThyssenKrupp dataset (c) Third sub time series from ThyssenKrupp dataset (d) Fourth sub time series from ThyssenKrupp dataset (e) Fifth sub time series from ThyssenKrupp dataset

Figure 6.13: Sub time series generated by splitting the original from ThyssenKrupp dataset



(a) Original spectrogram



(b) Convoluted spectrogram

Figure 6.14: Example of original and convoluted spectrogram based on ThyssenKrupp data

is from a good part of the BNA, while the corresponding final product is defect because other parts of the same BNA are the defect. However, after the preprocessing the good part has the same label as the defect parts, which seems to contradict its actual quality. One remark is that the label in our ThyssenKrupp data represents the quality of the final product but not the quality of the BNA.

6.2.2 Time-delay-embedding method

Similar to the experiments using standard dataset, we use two different inputs for Wasserstein distance calculation. The first is based on the time domain and the second is based on the frequency domain.

Based on time domain

We choose $k = 20$ in this experiment. After obtaining the Wasserstein distance matrix, we use Multidimensional scaling (MDS) with the desired dimension of 2. The result of learned representation is shown in figure 6.15. The orange points represent the good final product,

which means that its corresponding BNA is also good. We call it positive data in PU data scenario. The black points represent the defect final product. Its corresponding BNA quality is unknow. We call it unlabel data in PU data scenario.

We also calculate the proportion of the black points among the points from the periphery to the center of the cluster. During the preprocessing step, we split the original time series into 5 sub time series. It is possible that the points representing the sub time series coming from the same original are located differently because each part of the BNA has different quality. To handle this problem, we have two plots. One plot shows the proportion among the original time series. We calculate the proportion based on the original time series using algorithm 1.

Algorithm 1 Calculate proportion of black points among original time series

1: **Inputs:**
2: L_{sub} ▷ indices of the sub time series from the periphery to the center
3: **Initialize:**
4: $P = [\,]$ ▷ empty list for saving proportions
5: **for** n in range(1, 28080) **do**
6: $S \leftarrow \{\,\}$ ▷ Initialize set S for saving original time series indices
7: $x_1, \dots x_n \leftarrow L_{sub}[1], \dots, L_{sub}[n]$ ▷ Take from 1st to nth index from L_{sub}.
8: $S \leftarrow \left\lfloor \frac{x_1}{5} \right\rfloor, \dots, \left\lfloor \frac{x_n}{5} \right\rfloor$ ▷ Convert the sub time series indices back to indices of the original and save them *without repeats*
9: $n_{total} \leftarrow$ the number of elements in S
10: $n_{black} \leftarrow$ the number of elements in S that correspond to defect final product
11: $P \leftarrow \frac{n_{black}}{n_{total}}$ ▷ Append proportion to list P
12: **return** P

One remark is that in algorithm 1, some of the original time series indices $\lfloor \frac{x_i}{5} \rfloor$ obtained in step 8 might be the same because some sub time series can be from the same original. But set S only saves the unique indices. So the maximal number of points that can be included in S is 5616, which is the total number of the time series. And another shows the proportion among the sub time series. In this case, we calculate the proportion based on the sub time series. So the maximal number of counted points is $5616 * 5 = 28080$. The two plots are shown in figure 6.16.

From figures in 6.16, we see that the proportion of the black is high at the periphery of the point cloud. As more and more points are counted, the proportion finally drops to the ground truth proportion 0.03, which means that only a few orange points are at the periphery. So the majority at the periphery are the black points. It is highly possible that the black points at the periphery correspond to the defect part of BNA.

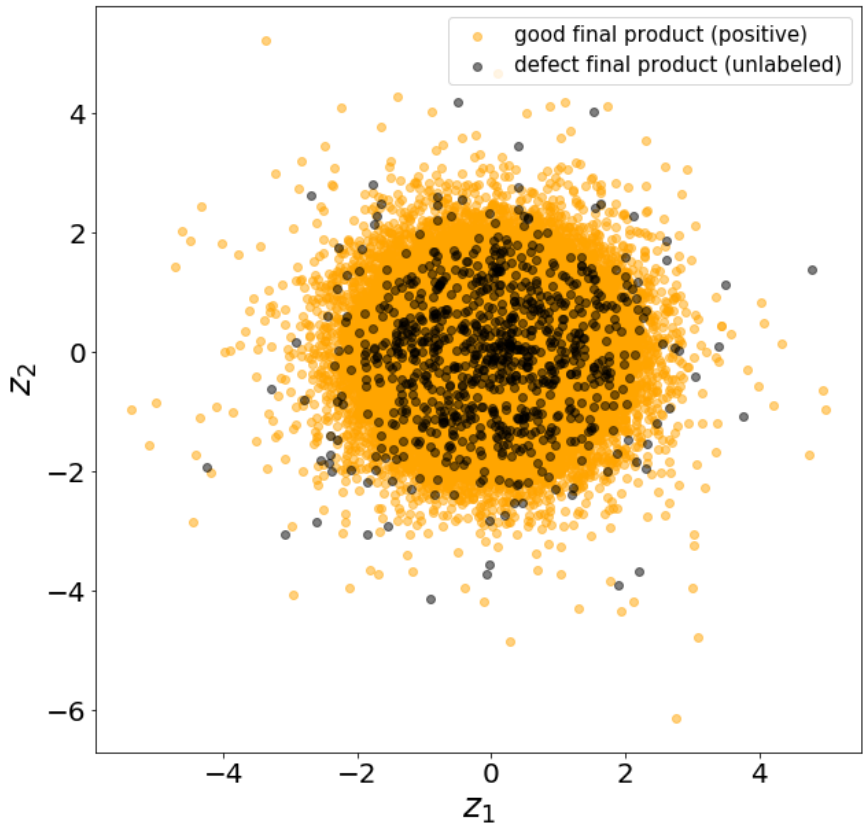
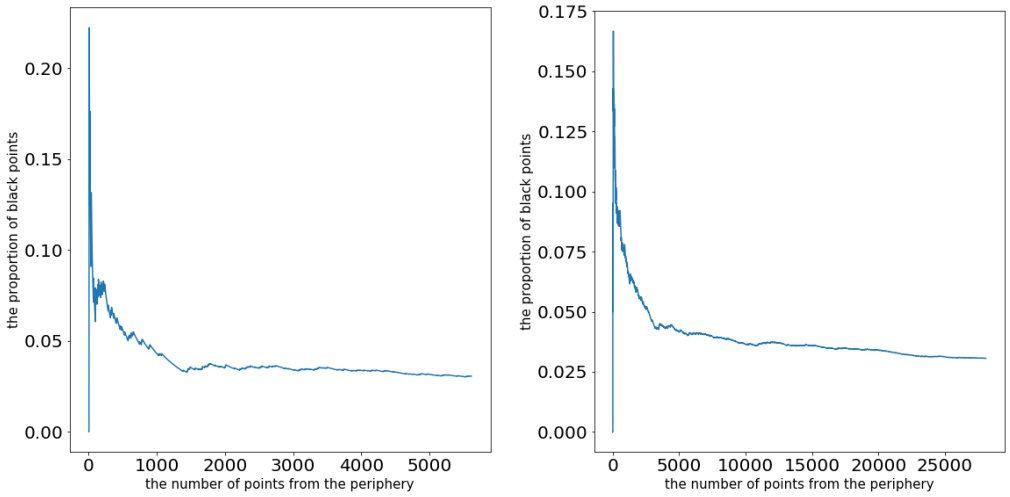


Figure 6.15: Learned representation of ThyssenKrupp data based on time domain



(a) Proportion among original time series based on time domain

(b) Proportion among sub time series based on time domain

Figure 6.16: Proportion of black points from periphery to center based on time domain using ThyssenKrupp dataset

Based on frequency domain

We use the convoluted spectrogram as the input and calculate the Wasserstein distance. The representation is obtained by multidimensional scaling. The result is shown in figure 6.17. The coloring scheme remains the same as we do in experiments based on the time domain. The orange stands for positive data, which means the final product is good. The black stands for unlabel data, which means the final product is defect.

Similarly, we plot the proportion of the black points for the spectrogram based on the original time series and based on the sub time series. They are shown in figure 6.18. The changing tendency of the proportion is similar to the result based on time domain information, shown in figure 6.16.

Then we check the points from the periphery and from the center in the point cloud. The points at the periphery have a large amplitude at certain frequency ranges in the corresponding convoluted spectrogram comparing with the points at the center. Some examples are shown in figure 6.19. The blue points in the point cloud represent the sampled points.

Another experiment we do is to use L2 distance instead of Wasserstein distance to measure the dissimilarity between each spectrogram. The result based on L_2 distance is shown in figure 6.20.

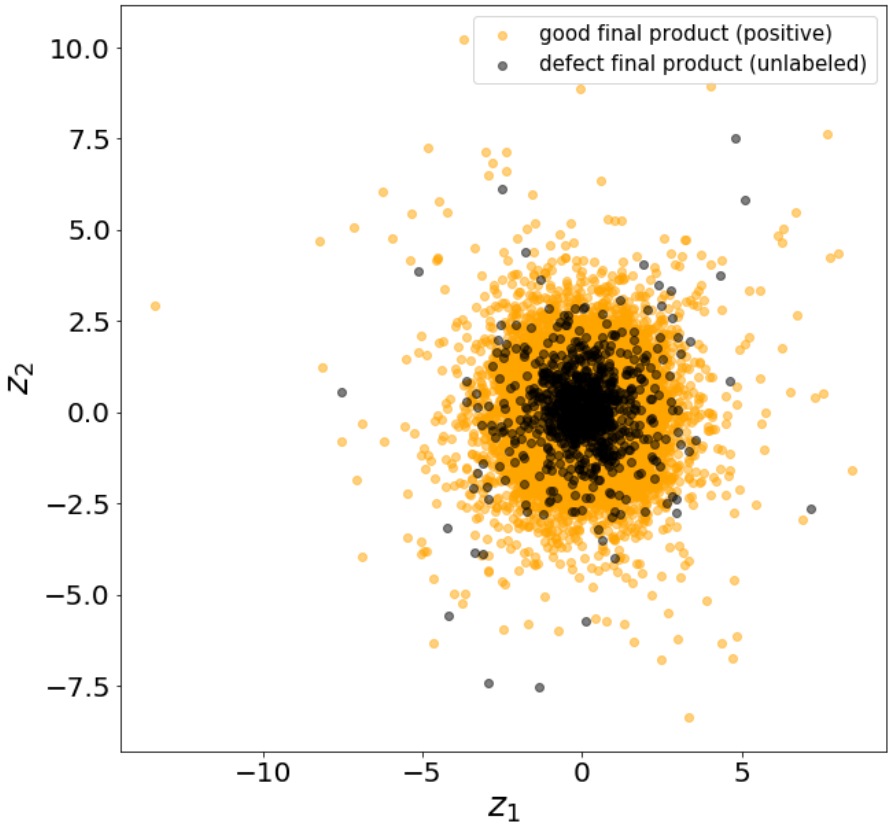


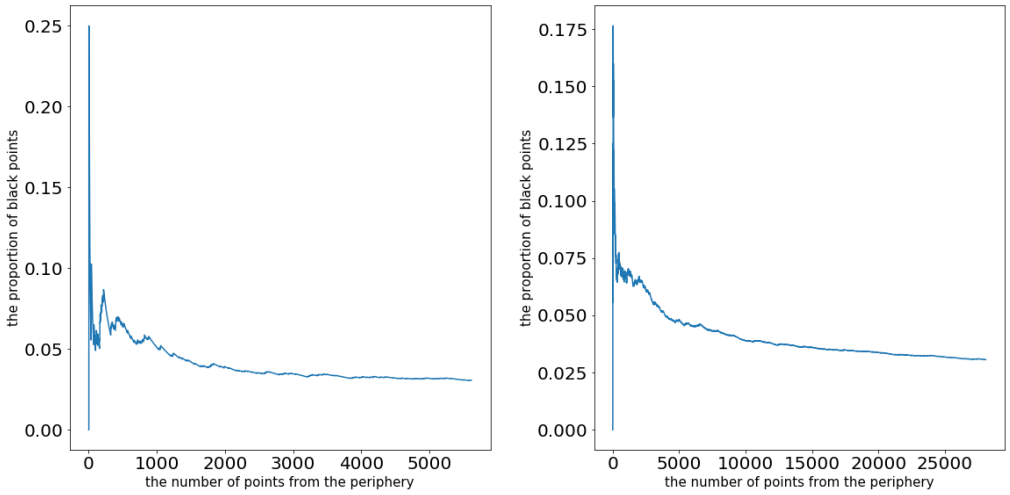
Figure 6.17: Learned representation of ThyssenKrupp data based on frequency domain

6.2.3 VAE method

Figure 6.21 is the representation learned by our VAE, which has the same architecture as we use in MNIST dataset experiments in the previous section.

6.2.4 SMI method

We choose 2 hidden layers with 500 and 20 neurons for representation learning and 1 hidden layer with 10 neurons for classification. The learned representations are shown in figure 6.22. The generated point cloud shows that most of the black points representing defect final product,



(a) Proportion among original time series based on frequency domain

(b) Proportion among sub time series based on frequency domain

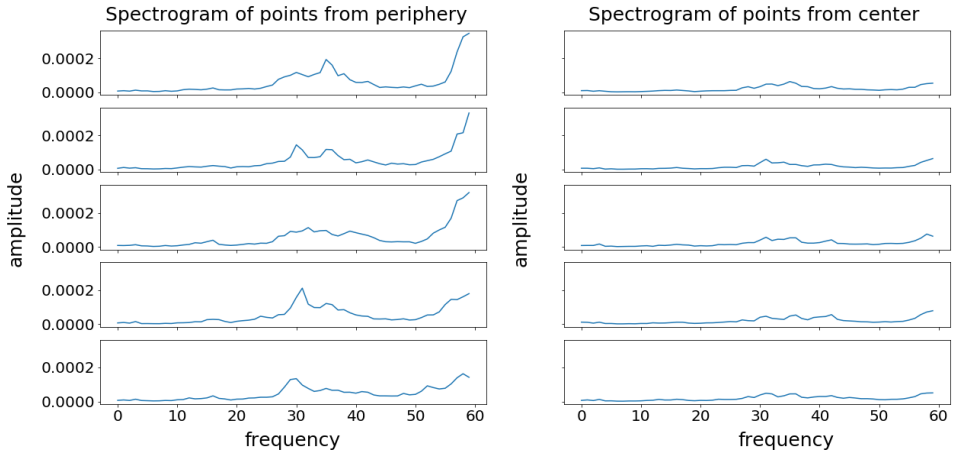
Figure 6.18: Proportion of black points from periphery to center based on frequency domain using ThyssenKrupp dataset

namely the unlabel data, are located on axis $z_2 = 0$.

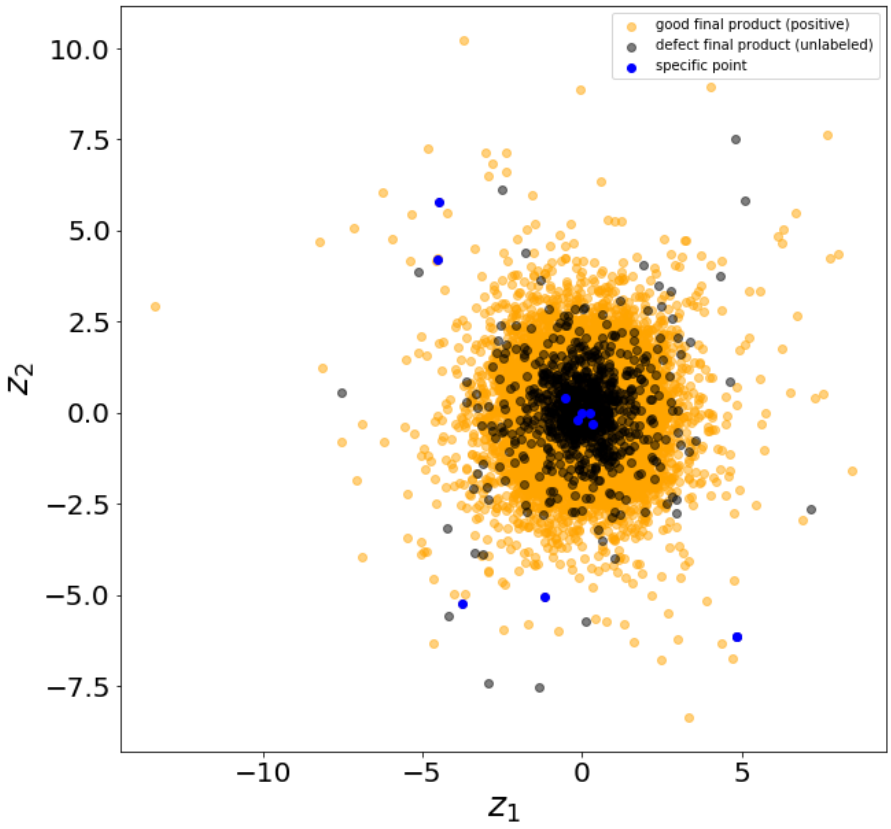
6.3 Consistency in different results

Based on the first method using frequency domain information, we see that the points at the periphery correspond to the spectrograms with high amplitude jump at some certain frequency interval. We take these points and check where they are located in the point cloud generated by L2 distance, VAE, and SMI method. The result is shown in figure 6.23.

The points from the periphery of the point cloud generated by Wasserstein distance are also at the periphery of the point cloud by using L2 distance. In the point cloud generated by VAE, these points are located at the tail of the two clusters. In the point cloud generated by SMI method, these points are located at axis $z_2 = 0$. So different methods learn the same feature in the input and present it differently. This consistency proves our methods capability of learning.



(a) Spectrogram of sampled points



(b) Locations of sampled points

Figure 6.19: Spectrograms of sampled points at center and at periphery using ThyssenKrupp dataset

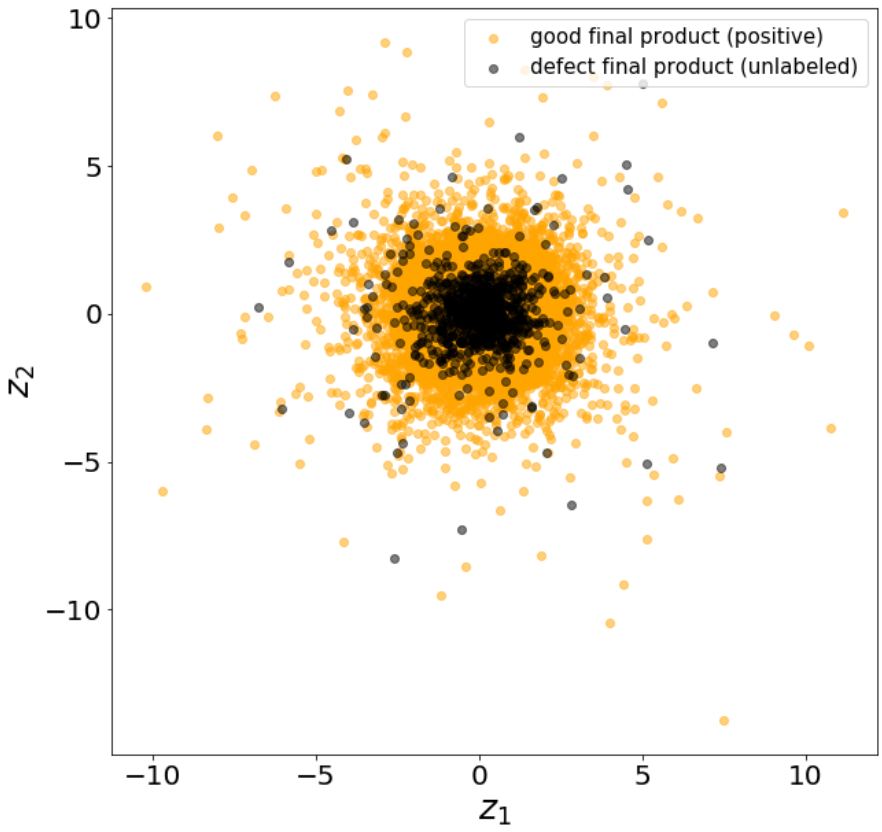


Figure 6.20: Learned representation of ThyssenKrupp data based on frequency domain using L_2 distance

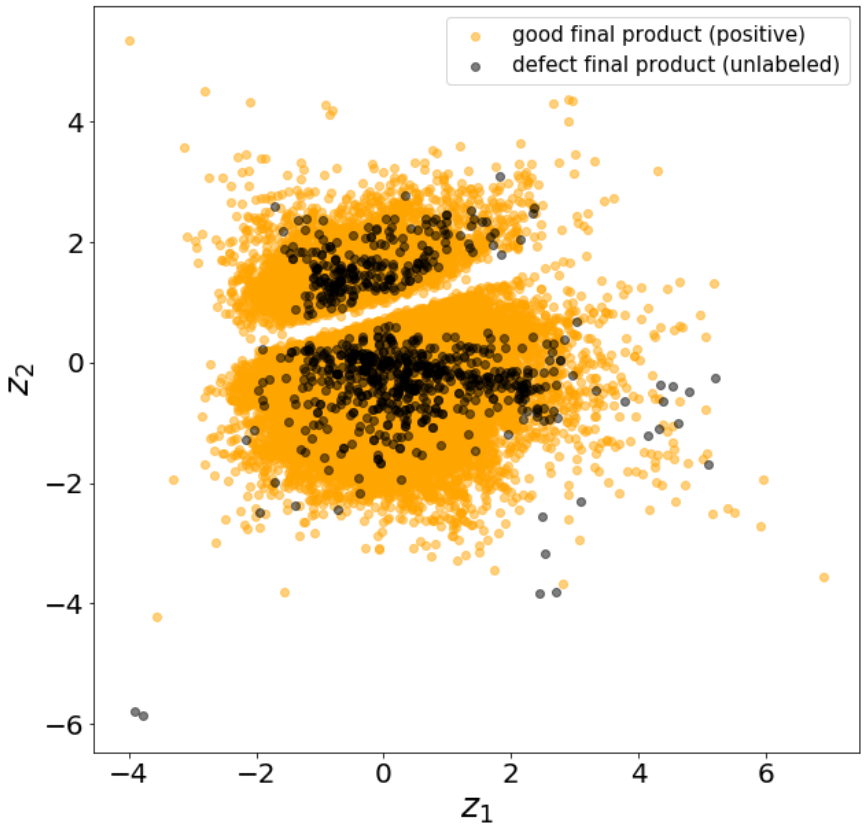


Figure 6.21: Representation of convoluted frequency data from ThyssenKrupp using VAE

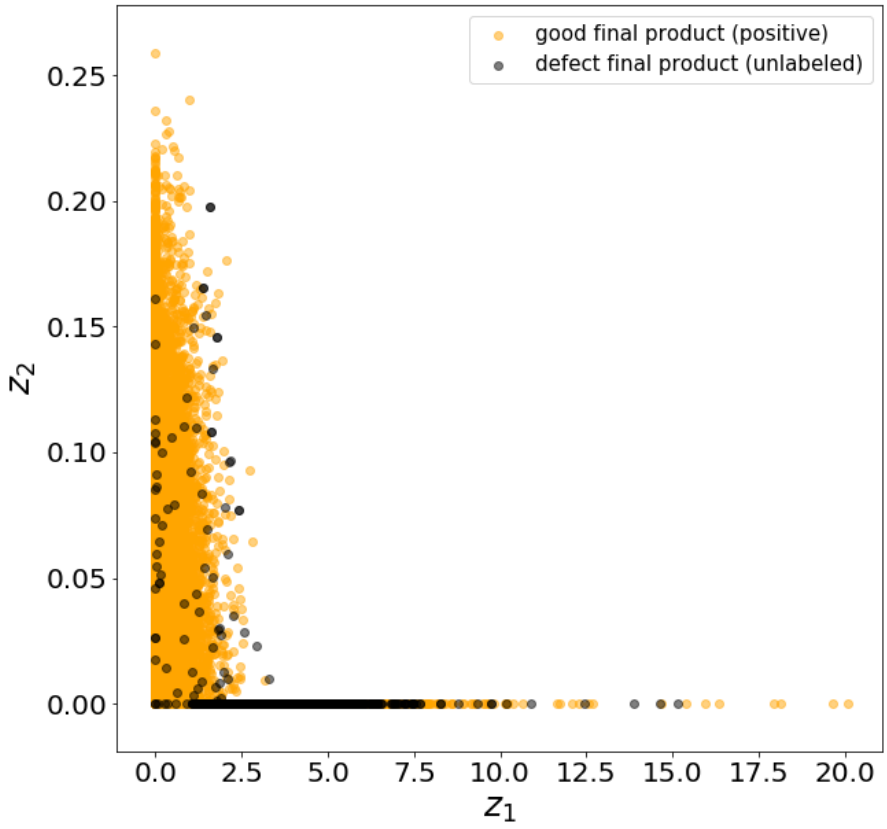
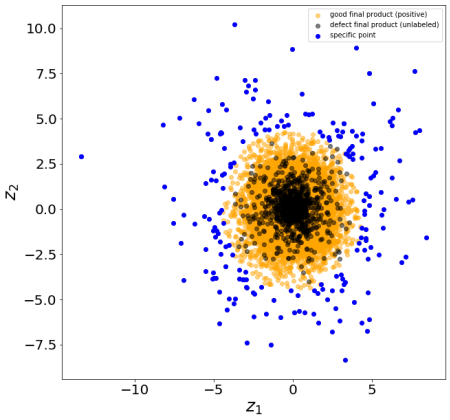
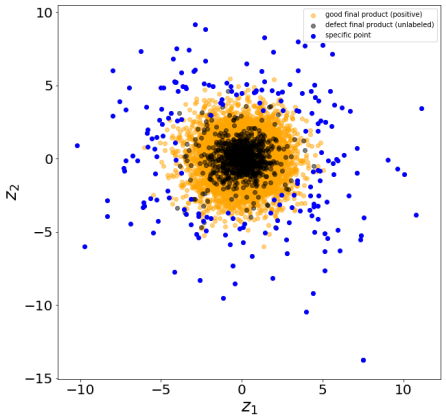


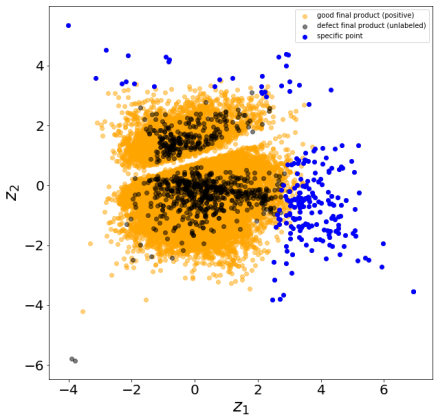
Figure 6.22: Learned representation of convoluted frequency data from ThyssenKrupp using SMI



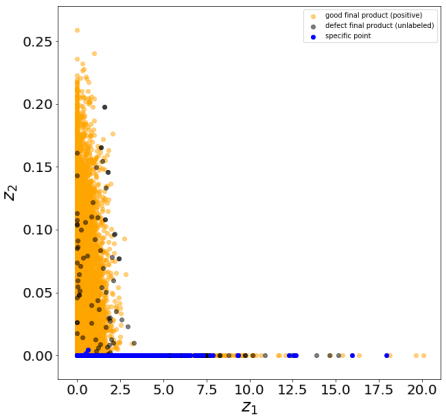
(a) Representation using spectrogram and Wasserstein distance



(b) Location of the blue points from figure 6.23a in representation using L_2 distance



(c) Location of the blue points from figure 6.23a in representation using VAE



(d) Location of the blue points from figure 6.23a in representation using SMI

Figure 6.23: Consistency of reprensetations in different methods using ThyssenKrupp dataset

Chapter 7

Summary and further works

In this thesis, we propose three different types of methods to solve the task from ThyssenKrupp, which is the representation learning of the signals generated by the rotation of BNA, a product made by ThyssenKrupp in Germany. The learned representation $\mathbf{z}$ is supposed to reveal the relationship between the quality of the final product steering gear, which is known label data and the quality of the BNA, which is unknown.

The first method we propose is based on time delay embedding, Wasserstein distance and multidimensional scaling (MDS). The idea of time delay embedding comes from dynamical system analysis. One example of using this method is to reconstruct the attractor. Here we interpret the result from time delay embedding to probability measure using delta function. Then we measure the difference between each probability measure using Wasserstein distance and find the representation with the desired dimension by MDS.

Time delayed embedding method is purely based on the time domain information. We also use real Fourier transform to get the spectrogram from the time series. The measure is then defined based on the spectrogram. The rest of the steps are the same.

We use NSynth dataset at first and then the dataset from ThyssenKrupp. The experiment shows one certain area of learned representations, whose corresponding spectrograms have a large amplitude in some certain interval of frequency.

The second method is variational autoencoder (VAE), which is based on advanced deep learning. We firstly use MNIST dataset to see the VAE's capability of finding representation in greatly reduced dimension. We also use VAE to generate new hand-written digits, which don't exist in the training or test datasets. After making sure that the implementation is correct, we used the spectrogram of the signal as input to train our VAE. The learned representation from VAE shows certain consistency with the result from the previous method, which is that the learned representation of spectrogram with huge jumps is located at the tail of the point

cloud. So both methods extract the same feature of the input data and represent them in the latent space differently.

The third method is the SMI method. It is a method based on information theory and applied to the positive unlabel data situation. Positive unlabel data means that the training set and the test set are composed of data with the positive data and unlabeled data, which is a mixture of positive data and negative data. This scenario perfectly fits the dataset given by ThyssenKrupp. In our case, the positive data is the good final product. We are sure that every component of steering gear is good in this case. And so it is for the component BNA. The unlabel data is the defect final product since we don't know whether it is caused by the BNA or not. It is composed of positive data (good BNA) and negative data (defect BNA). SMI model learns the representation and makes the prediction based on the learned representation.

The experiments based on MNIST dataset show that the SMI method is able to learn the representation of the unlabel data in the lower dimensional space efficiently and classify whether the unlabel data is from the positive data or from the negative data. It also produces the prediction of the mixing coefficient of the positive data among the unlabel data. The result of the SMI method using ThyssenKrupp data shows two clusters. The learned representations of defect final products are mostly located at axis $z_2 = 0$.

We find consistency in the results generated by these three methods. In experiments based on the first two methods, the points of the spectrogram with high amplitude are located either at the periphery of the only cluster or at the tails of the two clusters. In the point cloud generated by the SMI method, those points are now mostly located horizontally.

Because of the limited computation power, we don't try many tuning techniques for the hyperparameters of the neural network. Further studies include trying different hyperparameters to check the performance of the proposed methods. Another possible approach is to add a classifier to the VAE model and use the joint loss to train the encoder, decoder, and classifier simultaneously. This might improve the performance of the representation learning process and also directly gives us the prediction about the quality of the BNA. In general, this task from ThyssenKrupp is a combination of time series representation learning and semi-supervised learning. Because only the given label data of good final product provides information for the task. The label data of defect final products cannot provide as much information as the label of the good ones. More semi-supervise learning techniques are another main focus in further studies.

Bibliography

[1] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012.

[2] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[3] Geoff Boeing. Visual analysis of nonlinear dynamical systems: chaos, fractals, self-similarity and the limits of prediction. *Systems*, 4(4):37, 2016.

[4] Martin Casdagli, Stephen Eubank, J Doyne Farmer, and John Gibson. State space reconstruction in the presence of noise. *Physica D: Nonlinear Phenomena*, 51(1-3):52–98, 1991.

[5] Kin-Pong Chan and Wai-Chee Fu. Efficient time series matching by wavelets. In *icde*, page 126. IEEE, 1999.

[6] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[7] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi. Neural audio synthesis of musical notes with wavenet autoencoders, 2017.

[8] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. *SIGMOD Rec.*, 23(2):419–429, May 1994.

[9] Marco Giunti and Claudio Mazzola. Dynamical systems on monoids: Toward a general theory of deterministic systems and motion. In *Methods, models, simulations and approaches towards a general theory of change*, pages 173–185. World Scientific, 2012.

[10] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2017.

[11] I. Borg P. J. F. Groenen. *Modern Multidimensional Scaling*. Springer-Verlag New York, 2 edition, 2005.

[12] Holger Kantz and Thomas Schreiber. *Nonlinear Time Series Analysis*. Cambridge University Press, 2 edition, 2003.

[13] Andrej Karpathy, Justin Johnson, and Fei-Fei Li. Visualizing and understanding recurrent networks. *CoRR*, abs/1506.02078, 2015.

[14] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[15] Trupti M Kodinariya and Prashant R Makwana. Review on determining number of cluster in k-means clustering. *International Journal*, 1(6):90–95, 2013.

[16] Martin Längkvist, Lars Karlsson, and Amy Loutfi. A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, 42(10):11–24, 2014.

[17] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[18] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[19] Michael Muskulus and Sjoerd Verduyn-Lunel. Wasserstein distances in the analysis of time series and dynamical systems. *Physica D: Nonlinear Phenomena*, 240(1):45–58, 2011.

[20] Alex Nanopoulos, Rob Alcock, and Yannis Manolopoulos. Feature-based classification of time-series data. *International Journal of Computer Research*, 10:49–61, 2001.

[21] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[22] Haşim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *arXiv preprint arXiv:1402.1128*, 2014.

[23] Tomoya Sakai, Gang Niu, and Masashi Sugiyama. Estimation of squared-loss mutual information from positive and unlabeled data. *arXiv preprint arXiv:1710.05359*, 2017.

[24] Stephen Strogatz. *Nonlinear Dynamics And Chaos: With Applications To Physics, Biology, Chemistry, And Engineering (Studies in Nonlinearity)*. Westview Press, 2001.

[25] Masashi Sugiyama. Machine learning with squared-loss mutual information. *Entropy*, 15(1):80–112, 2012.

[26] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*, 2012.

[27] Floris Takens. Detecting strange attractors in turbulence. In *Dynamical systems and turbulence, Warwick 1980*, pages 366–381. Springer, 1981.

[28] Jianbo Yang, Minh Nhut Nguyen, Phyo Phyo San, Xiaoli Li, and Shonali Krishnaswamy. Deep convolutional neural networks on multichannel time series for human activity recognition. In *Ijcai*, volume 15, pages 3995–4001, 2015.