Computer Science and Information Engineering

National Chung Cheng University

Master's Thesis

Multi-domain generative adversarial network learning for time-series data generation

Student: Min-Shuin Tsai

Advisor: Dr. Chen-Kuo Chiang

Aug, 2021

Abstract

Time-series data generally refers to information with a time dimension, such as weather, expenditure, and even stock trends. These different types of data are all over our lives and profoundly affect our behavior. The prediction of time-series data is also an essential task in the field of deep learning. But time-series data needs to be recorded over time, which means it is difficult to collect enough data in a short time. The lack of information is also one of the problems in the field of deep learning. Therefore, we designed a framework based on *Time-Series Generative Adversarial Network* [1]. This method constrains a variety of fields to ensure the stability of the model, reduce the impact of model collapse and improve the quality of the generated data. The experimental results show that by training simple classification and prediction models, our method performs better than models without data augmentation and models with general data augmentation.

Keyword: Time Series, Generative Adversarial Network, Stability.

Contents

List of Tables

List of Figures

1 Introduction

Time-series data generally refers to information with a time dimension. For example, people's expected weather forecasts, consumption records, and even market stock price trends can all be counted as time-series data. These data are all over our lives. , And unknowingly affect people's lives. In addition to time-series data with time records, another noticeable feature is that the current data is often affected by the first few, or even the first tens or hundreds of data. In other words, the conditions that determine the status of the current data often depend on the previous data. Because of this, in the field of deep learning, compared with common image recognition, positioning, and other tasks, time-series data has been extended to time-series data unique tasks, such as time-series prediction [2], time-series anomaly detection [3] and additional charges. But in deep learning, task success often comes from having a lot of information for training. The time-series data with a time dimension means that its data collection must be recorded over time. In other words, it is difficult for us to collect the time-series data required for training in a short time. Because of the difficulty of collecting time-series data and its time dependence, it isn't easy to apply standard image amplification methods to time-series data. In recent years, researchers have also proposed various ways to generate realistic synthetic data.

With technological methods in recent years, data generation methods for time-series data have gradually increased. In the early data generation methods, in addition to the most primitive method of directly producing data in time domain [4] [5] [6] [7], other methods convert the data to frequency domain [8] [9] to generate data. With the development of technology and the study of the characteristics of time-series data, [10] proposed a learning-based method. And [11] investigates and consolidates the time-series data generation methods and believes that the combination of different ways can get better results. Among the learning-based techniques, an approach based on generative adversarial network [12] [13–15] has also been proposed. [1] believes that these methods can not easy to learn the relationship between the characteristics of data and the underlying time dynamics by directly producing time-series data through generators. [1] believes that data can show its features and temporal dynamics more clearly in high-dimensional space. Therefore, by mapping data to a high-dimensional area, [1] hopes that the generator can learn the characteristics of data better by directly producing the distribution of data in a high-dimensional space. The method in this article is based on the idea of [1], and further expands the data to other domains, and strengthens the learning ability of the model by learning the distribution and time characteristics of data in different domains.

The method in this paper is based on the concept of [1]. In addition to focusing on training in high-dimensional space, further training focuses on the domain of data and noise, strengthening the connection between data in different environments. Let the model can learn the characteristics and time dynamics of data more thoroughly. We designed two sets of auto-encoders to strengthen the connection between data, latent code, and noise. In addition, we use unsupervised learning to make synthetic latent noise approximate the distribution of actual noise. Finally, we use the data discriminator to compare the distribution of the final generated data with the real data to ensure that the distribution of the generated data is similar to the original data.

This paper has three contributions:

1. We propose a time-series data augmentation method, which can better learn the feature representation of the data in latent space.

2. Our method can better learn the accuracy of the distribution of the real data.

3. Our method generates the synthetic data and mixes it with the real data for training can improve the model's performance.

2 Related Works

Here we will introduce common time-series data augmentation methods, learning-based methods, and GAN-based methods.

2.1 Survey of Time-series Data Generation Methods

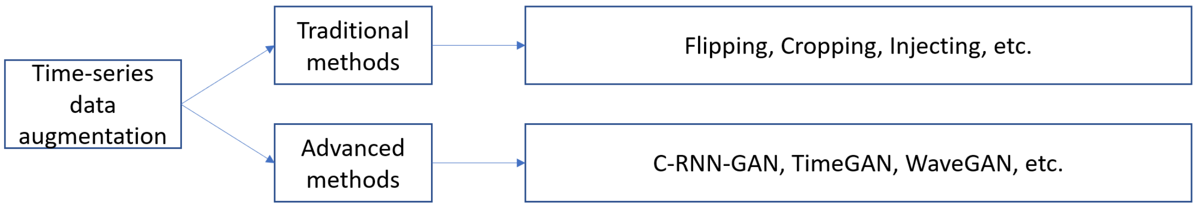

Time-series data augmentation
Traditional methods
Advanced methods
Flipping, Cropping, Injecting, etc.
C-RNN-GAN, TimeGAN, WaveGAN, etc.

Figure 1: A classification of time-series data augmentation methods. Traditional methods mostly add data by modifying the original data, while the advanced methods primarily generate data by learning the feature representation of the original data.

We investigated the existing time-series data generation methods and divided them into traditional methods and advanced methods, as shown in Figure 1. Traditional methods primarily amplify data by operating on time-series data. [4] proposes a way that can flip a time-series to get a new time-series which is regarded as the original data. [5] introduces window cropping, which is a method of randomly extracting consecutive slices from the original time series to obtain sub-samples, where the length of the slice is an adjustable parameter. [6] forms new data without changing the corresponding label by injecting noise or abnormal values into the time-series data. Noise or outliers that can be injected include spikes, Gaussian noise, step-like trends, etc. [7] uses Dynamic Time Wrapping(DTW) [16, 17] and DTW Barycenter Averaging [16] (DBA) to find the average time series, and generates a new time series by adding disturbances on it. [8] increases the data by converting the data from the time domain to the frequency domain and adding disturbances to the amplitude spectrum values. [9] adds disturbance to the data converted to the frequency domain and then converts it back to the time domain to augment the original data.

2.2 Learning-based Methods

In learning-based methods, they mainly learn the features in time-series data and generate new time-series. [10] proposes to learn and perform data enhancement in features space. [10] chooses to use auto encoder [18] to convert data to feature space and generate new samples through interpolation and extrapolation. Variational autoencoder (VAE) [19] uses an autoencoder to convert data into low-dimensional vectors and then reconstructs them. The difference between [10] and [19] is that they generate new data by adding noise to low-dimensional vectors and reconstructing them. Wavenet [14] uses temporal convolution network [20] to increase the sensitivity to long periods and is used to generate audio. Since [12] proposed the generative adversarial network (GAN), it has become a prevalent task in the learning method. GAN conducts adversarial learning between generators and discriminators. In the end, the generator can generate synthetic data that is indistinguishable from actual data, and therefore extends many applications [13,15,21–25] and improvements [26–28].

2.3 GAN-based Methods

The mechanism of the generative adversarial network allows it to be used for generation tasks in various fields. Among them, C-RNN-GAN [13] first uses the autoregressive model on GAN and uses it to generate time-series data. WaveGAN [15] uses DCGAN [29] as the framework and applies GAN to audio tasks. VAE GAN [21] uses a discriminator based on VAE [19] to distinguish the reconstructed data from the original data to improve the similarity. Bidirectional GAN [22] trains the encoder and decoder separately and uses a discriminator to differentiate the feature and noise generated by the encoder so that the feature caused by the encoder can be similar to noise and achieve using noise to create data. Adversarial autoencoder [23] also hopes that the latent vector of the data can be similar to noise, so the discriminator is used to distinguish the latent vector generated by the encoder and noise and achieves using noise to create new data. Recurrent conditional GAN [24] adds control conditions to GAN to control the types of the generated data. Geometric GAN [25] geometrically reinterprets GAN and proposes a new loss function called hinge loss, which uses the hinge loss function in [30,31], and this paper also uses this loss function. TimeGAN [1] believes these current methods learn the feature representation of data by directly generating data through the noise isn't easy to understand the underlying timing dynamics of the data and the relationship between multiple features. TimeGAN believes that time-series can fully express its characteristics in high-dimensional space. Therefore, when latent code is generated through auto-encoder, the generator is changed to generate latent code to learn data distribution in high-dimensional space better. At the same time, TimeGAN also performs supervised training and adversarial training on real latent code and synthetic latent code to ensure accurate learning characteristics and distribution.

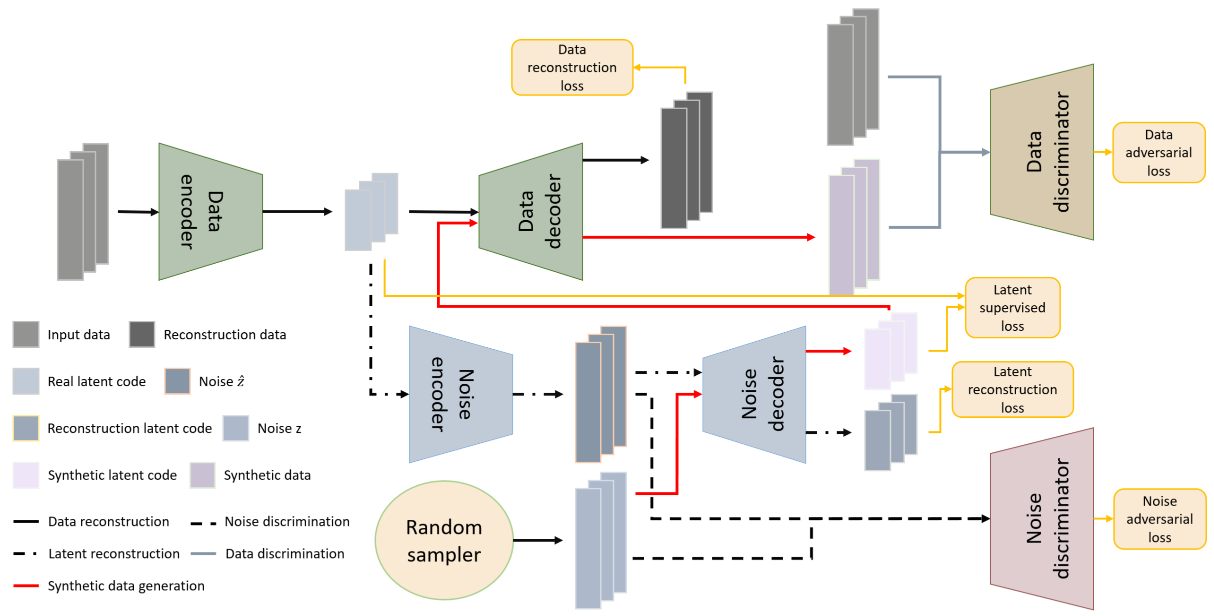
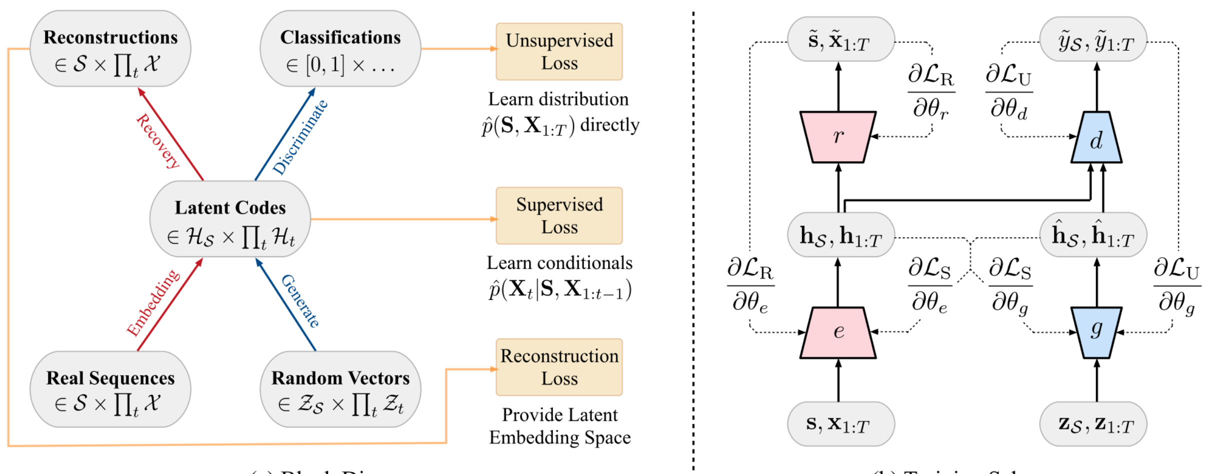


Figure 2: The system framework of our method. It is divided into three parts. The first part is the data auto-decoder part in the upper left corner. This part mainly embeds the input data to the latent space, receives the latent code, and recovers it to the data domain. The second part is the noise auto-decoder in the lower-left corner, which mainly embeds the real latent code to the noise domain, and recovers the synthetic noise and to the latent space. The last part is discriminators, which are responsible for the identification of data and noise domains.

3 Proposed Method

In this paper, our goal is to train a time-series data augmentation model to generate synthetic data similar to real data for the purpose of generating time-series data. In section 3.1, we will introduce Time-series generative adversarial network [1]. In section 3.2, we will introduce the architecture of our method. In section 3.3, we will introduce each part of the method individually.



Reconstructions $\in \mathcal{S} \times \prod_t \mathcal{X}$
Classifications $\in [0,1] \times \ldots$
Unsupervised Loss
Learn distribution $\hat{p}(\mathbf{S}, \mathbf{X}_{1:T})$ directly
Recovery
Discriminate
Latent Codes $\in \mathcal{H}_{\mathcal{S}} \times \prod_t \mathcal{H}_t$
Supervised Loss
Learn conditionals $\hat{p}(\mathbf{X}_t|\mathbf{S}, \mathbf{X}_{1:t-1})$
Embedding
Generate
Real Sequences $\in \mathcal{S} \times \prod_t \mathcal{X}$
Random Vectors $\in \mathcal{Z}_{\mathcal{S}} \times \prod_t \mathcal{Z}_t$
Reconstruction Loss
Provide Latent Embedding Space
$\tilde{\mathbf{s}}, \tilde{\mathbf{x}}_{1:T}$
$\tilde{y}_{\mathcal{S}}, \tilde{y}_{1:T}$
$\frac{\partial \mathcal{L}_{\mathrm{R}}}{\partial \theta_r}$
$\frac{\partial \mathcal{L}_{\mathrm{U}}}{\partial \theta_d}$
r
d
$\mathbf{h}_{\mathcal{S}}, \mathbf{h}_{1:T}$
$\hat{\mathbf{h}}_{\mathcal{S}}, \hat{\mathbf{h}}_{1:T}$
$\frac{\partial \mathcal{L}_{\mathrm{R}}}{\partial \theta_e}$
$\frac{\partial \mathcal{L}_{\mathrm{S}}}{\partial \theta_e}$
$\frac{\partial \mathcal{L}_{\mathrm{S}}}{\partial \theta_g}$
$\frac{\partial \mathcal{L}_{\mathrm{U}}}{\partial \theta_g}$
e
g
$\mathbf{s}, \mathbf{x}_{1:T}$
$\mathbf{z}_{\mathcal{S}}, \mathbf{z}_{1:T}$

(a) Block Diagram (b) Training Scheme

Figure 3: Diagram and scheme of TimeGAN method. Part (a) is a schematic diagram of the architecture. The generator in the figure produces synthetic latent code. (b) is the training process of the method, which combines unsupervised and supervised loss to train the generator. Source: [1].

3.1 Time-series Generative Adversarial Network

TimeGAN [1] is a paper published in NeurIPS 2019. It believes that data can better show the relationship between its distribution and features in high-dimensional space, so it designs a generative adversarial network to train the data in high-dimensional space, such as Figure 3 as shown. (a) is divided into two parts, the autoencoder part on the left and the generator and discriminator parts on the right. First, explain the part of the autoencoder on the left. The autoencoder on the left obtains the distribution of the data $\hat{p}(\mathrm{S}, \mathrm{X}_{1:T})$ in the latent space $\mathcal{H}$. The generator on the right is different from the generator of the general method that directly generates data. Still, it generates the distribution of noise in the latent space, which means that the generator directly learns the more obvious complex correlation between the distribution and the feature in the high-dimensional space. The discriminator is responsible for distinguishing the latent code from the data embedded to the latent space and the synthetic latent code generated by the generator. At the same time, TimeGAN also conducts supervised training for real latent code and synthetic latent code to ensure that the model can capture the stepwise conditional distribution $\hat{p}(\mathrm{X}_t|\mathrm{S}, \mathrm{X}_{1:t-1})$ in the data. (b) is the training process of TimeGAN, where $\mathcal{L}_R$ is the reconstruction loss part of (a), $\mathcal{L}_S$ is supervised loss, $\mathcal{L}_U$ is unsupervised loss. The solid line is the forward part, and the dotted line is the backward part. We can see that the supervised loss in (b) optimizes the encoder and generator.

3.2 System Framework

Our architecture diagram is shown in Figure 2. First, introduce the notation in the method. Input data represents our actual dataset X. Noise z means our noise is randomly sampled from the normal distribution. $\hat{z}$ refers to synthetic noise. The solid line represents the reconstruction of the data, the other dotted lines from top to bottom, from left to the right represent the process of distinguishing real or synthetic noise, the reconstruction of latent, and the process of distinguishing natural or artificial data. The solid red line represents the process that finally generates synthetic data. Next, we will introduce our method and architecture.

The architecture of this paper is divided into four parts: data autoencoder, noise autoencoder, noise discriminator and data discriminator. First, we train the data autoencoder to input the real data and reconstruct it, and at the same time generate the distribution of data in the latent space. This training is a process of learning how to rebuild and generate the latent code applied to the input in next stage. Next, we input the generated latent code into the noise autoencoder to learn how to reconstruct it. The noise encoder maps the latent code to the noise domain and reconstructs it through the noise decoder. Then the noise discriminator will compare latent vector $\hat{z}$ embedded from latent code with noise z sampled from normal distribution to make the noise encoder generate synthetic noise similar to the real noise distribution. This part of the study hopes to strengthen the relationship between latent space and noise domain by learning how to reconstruct the latent code and real noise fitting process and use real noise to generate new data later. We can strengthen the three fields and generate synthetic data through real noise by combining the two autoencoders. We use the data discriminator to fit the synthetic data with real data to ensure further that our target can approximate actual data.

3.3 Parts of System Framework

In section 3.3, we will explain the four processes mentioned in section 3.2 separately and introduce their loss functions and how to optimize them.

3.3.1 Data Reconstruction

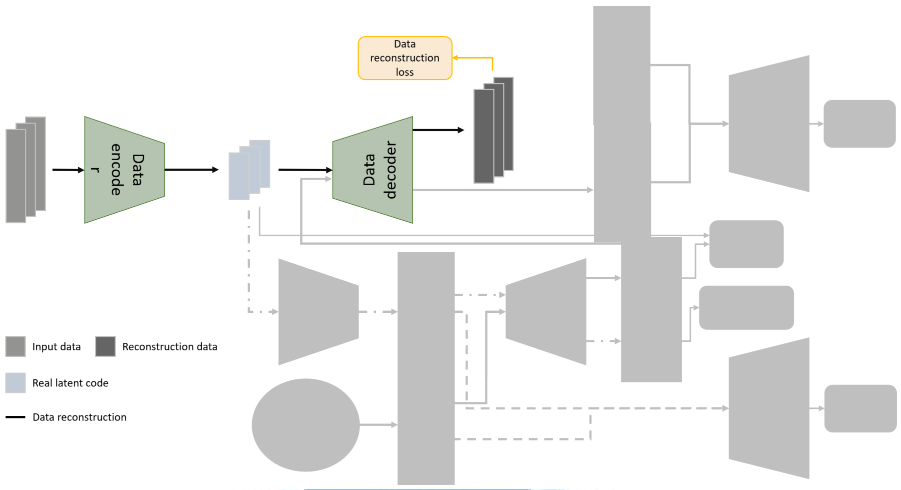


Figure 4: Data reconstruction. This figure shows the part of data reconstruction in the architecture diagram.

In the figure 4, we replace the input data with $x_{1:T}^{1:n}$, $x_{1:T}^{1:n}$ is sampled from real data X. Among them, n is the number of features and T is the number of time steps. For convenience, superscripts will be omitted from now on. We use $h_{1:T}$ to instead the real latent code. Reconstruction data is represented by $\tilde{x}_{1:T}$. The data reconstruction process uses the data encoder Dec_{data} to map $x_{1:T}^{1:n}$ to the latent space H, while using data decoder to recovery $h_{1:T}$ to $\tilde{x}_{1:T}$. In order to make the reconstructed $\tilde{x}_{1:T}$ approximate to $x_{1:T}$, our loss function is defined as follows:

$$\mathcal{L}_R^{data}(\theta_{Enc_{data}}, \theta_{Dec_{data}}) = \mathbb{E}_{x_{1:T}\sim p(X)}(\|x_{1:T} - Dec_{data}(Enc_{data}(x_{1:T}))\|_2) \tag{1}$$

3.3.2 Latent Reconstruction

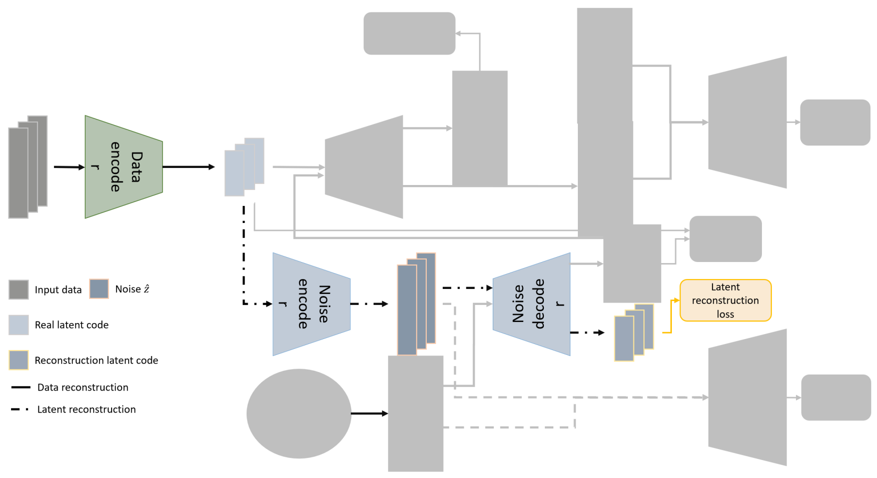


Figure 5: Latent reconstruction. The figure shows the part of latent reconstruction in the architecture diagram. The dotted line in the figure is the latent reconstruction forwarding process.

In the figure 5, we input $h_{1:T}$ embedded from Enc_{data} into noise autoencoder and reconstruct it into reconstruction latent code $\tilde{h}_{1:T}$. Among them, we use noise encoder Enc_{noise} to embed $h_{1:T}$ into noise domain to generate synthetic latent vector $\hat{z}_{1:T}$, and then use noise decoder Dec_{noise} returns $\hat{z}_{1:T}$ to reconstruction latent code $\tilde{h}_{1:T}$. In order to make the reconstructed $\tilde{h}_{1:T}$ approximate to $h_{1:T}$, our loss function is defined as follows:

$$\mathcal{L}_R^{latent}(\theta_{Enc_{noise}}, \theta_{Dec_{noise}}) = \mathbb{E}_{h_{1:T}\sim p(H)}(\|h_{1:T} - Dec_{noise}(Enc_{noise}(h_{1:T}))\|_2) \qquad (2)$$

3.3.3 Noise Discrimination

In the figure 5, we pass the $h_{1:T}$ generated by Enc_{data} and let Enc_{noise} embed it to noise domain Z to generate $\hat{z}_{1:T}$. But this is not enough to show that the mapping relationship between the two domains can be strengthened and used to generate new data. So we use noise discriminator $\mathcal{D}_{noise}$ to distinguish between $\hat{z}_{1:T}$ and nosie $z_{1:T}$, as shown in the figure 6. By disrupting the distribution of $h_{1:T}$ in the noise domain, $\hat{z}_{1:T}$ can approximate the distribution of $z_{1:T}$ and use Dec_{noise} to make the distribution similar to noise $\hat{z}_{1:T}$ rebuild into $\tilde{h}_{1:T}$. Then uses $z_{1:T}$ which sampled from normal distribution to generate new data. This way strengthens the mapping relationship between the two

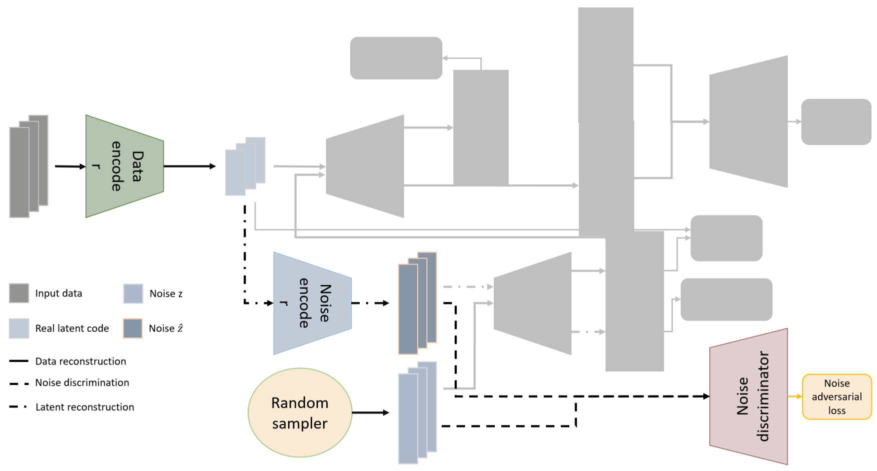


Figure 6: Noise discrimination. The figure shows the part of noise discrimination in the architecture diagram. The long dotted line in the figure is the process of noise discriminator distinguishing synthetic latent vector $\hat{z}_{1:T}$ and $z_{1:T}$.

domains. In order to make $\hat{z}_{1:T}$ approximate to $z_{1:T}$, we use adversarial training, where Enc_{noise} is regarded as a generator in this process, so the loss function is defined as follows:

$$\mathcal{L}_{\mathcal{G}}^{noise}(\theta_{Enc_{noise}}) = -\mathbb{E}_{h_{1:T}\sim p(H)}[\mathcal{D}_{noise}(Enc_{noise}(h_{1:T}))] \tag{3}$$

$$\begin{aligned}\mathcal{L}_{\mathcal{D}}^{noise}(\theta_{\mathcal{D}_{noise}}) =& \mathbb{E}_{z_{1:T}\sim p(Z)}[\max(0, 1-\mathcal{D}_{noise}(z_{1:T}))]+\\ & \mathbb{E}_{h_{1:T}\sim p(H)}[\max(0, 1+\mathcal{D}_{noise}(Enc_{noise}(h_{1:T})))]\end{aligned} \tag{4}$$

3.3.4 Data Discrimination

In the final process of generating synthetic data, we will divide it into two parts: supervised training and data discrimination. To further strengthen our goal, which is the synthetic data we need, we combine supervised training and unsupervised training so that our model can better capture the characteristics and distribution of data. Figure 7 respectively demonstrates the process of supervised training and unsupervised training. On the left is the part of supervised training. Enc_{data} maps $x_{1:T}$ to latent space to generate $h_{1:T}$, and Dec_{noise} uses $z_{1:T}$ to generate $\hat{h}_{1:T}$. To capture stepwise conditional distribution and ensure that the $\hat{h}_{1:T}$ generated by $z_{1:T}$ can approximate $h_{1:T}$ by supervised training,

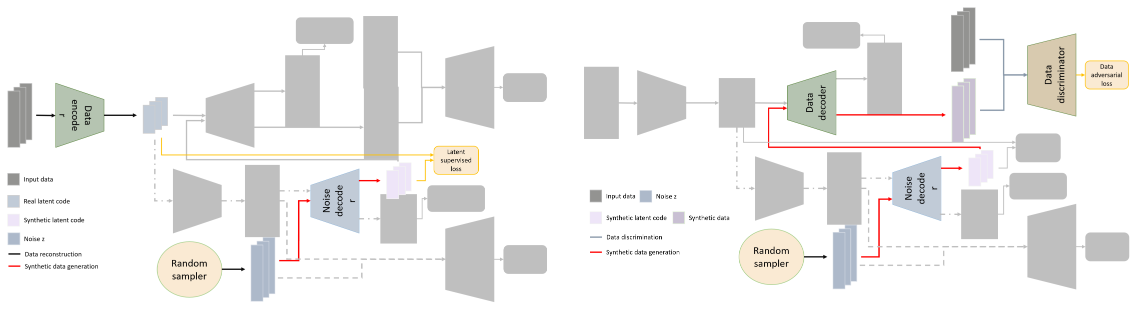


Figure 7: Supervised training and data discrimination.

The figure from left to right shows the process of supervised training and data discrimination. The figure on the left is mainly for supervised training of real and synthetic latent codes from noise. The data discriminator on the right compares the final generated data with the original data.

our supervised loss function is defined as follows:

$$\begin{aligned}&\mathcal{L}_S^{latent}(\theta_{Dec_{noise}}, \theta_{Dec_{data}}, \theta_{\mathcal{D}_{data}}) = \\ &\mathbb{E}_{x_{1:T}\sim p(X), z_{1:T}\sim p(Z)}(\|Enc_{data}(x_{1:T}) - Dec_{noise}(z_{1:T})\|_2)\end{aligned} \tag{5}$$

Finally, we use the data discriminator to conduct adversarial training on $\hat{x}_{1:T}$ generated by Enc_{data} and Dec_{data} and $x_{1:T}$, and finally ensure that the distribution of $\hat{x}_{1:T}$ and $x_{1:T}$ can be similar. Note that the generator here is a combination of Enc_{data} and Dec_{data}. Our data adversarial loss function is defined as follows:

$$\mathcal{L}_{\mathcal{G}}^{data}(\theta_{Dec_{noise}}, \theta_{Dec_{data}}) = -\mathbb{E}_{z_{1:T}\sim p(Z)}[\mathcal{D}_{data}(Dec_{data}(Dec_{noise}(z_{1:T})))] + \mathcal{L}_S^{latent} \tag{6}$$

$$\begin{aligned}\mathcal{L}_{\mathcal{D}}^{data}(\theta_{\mathcal{D}_{data}}) =&\mathbb{E}_{x_{1:T}\sim p(X)}[\max(0, 1 - \mathcal{D}_{data}(x_{1:T}))] + \\ &\mathbb{E}_{z_{1:T}\sim p(Z)}[\max(0, 1 + \mathcal{D}_{data}(Dec_{data}(Dec_{noise}(z_{1:T}))))] + \mathcal{L}_S^{latent}\end{aligned} \tag{7}$$

3.3.5 Total Loss

Finally, we integrate all processes with loss functions and optimize the overall architecture. Our total loss is as follows:

$$\begin{aligned}&\mathcal{L}_{total} = \\ &\lambda_R^{data}\mathcal{L}_R^{data}(\theta_{Enc_{data}}, \theta_{Dec_{data}}) + \lambda_R^{latent}\mathcal{L}_R^{latent}(\theta_{Enc_{noise}}, \theta_{Dec_{noise}}) + \\ &\lambda_{\mathcal{G}}^{noise}\mathcal{L}_{\mathcal{G}}^{noise}(\theta_{Enc_{noise}}) + \lambda_{\mathcal{D}}^{noise}\mathcal{L}_{\mathcal{D}}^{noise}(\theta_{\mathcal{D}_{noise}}) + \\ &\lambda_{\mathcal{G}}^{data}\mathcal{L}_{\mathcal{G}}^{data}(\theta_{Dec_{noise}}, \theta_{Dec_{data}}) + \lambda_{\mathcal{D}}^{data}\mathcal{L}_{\mathcal{D}}^{data}(\theta_{\mathcal{D}_{data}})\end{aligned} \tag{8}$$

Where λ_R^{data}, λ_R^{latent}, $\lambda_{\mathcal{G}}^{noise}$, $\lambda_{\mathcal{D}}^{noise}$, $\lambda_{\mathcal{G}}^{data}$, $\lambda_{\mathcal{D}}^{data}$ are our hyperparameters to balance the proportion between different losses.

4 Implementation

In this chapter, we will introduce some details of our method and show the flow of our approach.

First, the components of our method are composed of five layers of GRU [32] and a fully connected layer. During training, we set the batch size to 64. We use Adam [33] as the optimizer. We design the learning rate according to different datasets, but we halve the learning rate every 1/5 training cycles. In section 3.3.2, 3.3.3, 3.3.4, we will fix the parameters of the data encoder to treat the real latent code as the label. In sections 3.3.3 and 3.3.4, when the generator is training, we will fix the parameters of the discriminator; otherwise, we will fix the parameters of the generator. Note that the four processes in the method are all in the same iteration. That is to say, actual data and noise are all the same. In addition, in the part of discrimination, due to the need of the adversarial loss function, we set the positive sample as 1 and the negative sample as -1. Finally, the process of generating data requires the noise decoder and data decoder.

The following demonstrates the algorithm flow of our method. We also divide the algorithm into four processes, each reflecting the calculation of forwarding and loss function. At the same time, we will also explain the practice and significance of each process.

Algorithm 1 Adversarial Training For Time Series Data Generation

```
Require: training dataset $X$ and normalization distribution $Z$. $T$: the number of time steps.
Ensure: Generated data $\hat{X}$.
1: for each iteration training do
2:     Randomly sample $x_{1:T}$, $z_{1:T}$ from training dataset and normal distribution.
3:     Data reconstruction. We generate the corresponding latent code $\hat{h}_{1:T}$ and make the reconstructed data $\tilde{x}_{1:T}$ similar to the original data $x_{1:T}$. Compute $\mathcal{L}_R^{data}$ and optimize the data auto-encoder $(\theta_{Enc_{data}}, \theta_{Dec_{data}})$.
           $\tilde{x}_{1:T} = Dec_{data}(h_{1:T}), \quad h_{1:T} = Enc_{data}(x_{1:T}).$
           $\mathcal{L}_R^{data}(\theta_{Enc_{data}}, \theta_{Dec_{data}}) = \mathbb{E}_{x_{1:T}\sim p(X)}(\|x_{1:T} - Dec_{data}(Enc_{data}(x_{1:T}))\|_2).$
4:     Latent reconstruction. We generate the corresponding latent vector $\hat{z}_{1:T}$ and make the reconstructed latent code $\tilde{h}_{1:T}$ similar to the original latent code $h_{1:T}$. Com-
```

```
    pute $\mathcal{L}_R^{latent}$ and optimize the noise auto-encoder $(\theta_{Enc_{noise}}, \theta_{Dec_{noise}})$.
        $\tilde{h}_{1:T} = Dec_{noise}(\hat{z}_{1:T}), \quad \hat{z}_{1:T} = Enc_{noise}(h_{1:T}).$
        $\mathcal{L}_R^{latent}(\theta_{Enc_{noise}}, \theta_{Dec_{noise}}) = \mathbb{E}_{h_{1:T}\sim p(H)}(\|h_{1:T} - Dec_{noise}(Enc_{noise}(h_{1:T}))\|_2).$
5:  **Noise discrimination.** We use $\mathcal{D}_{noise}$ to discriminate $\hat{z}_{1:T}$ and $z_{1:T}$. We hope that generated noise can match the distribution of $z_{1:T}$, so that $z_{1:T}$ can be used to generate $\hat{h}_{1:T}$ later. Compute $\mathcal{L}_{\mathcal{G}}^{noise}$, $\mathcal{L}_{\mathcal{D}}^{noise}$ and optimize the noise encoder $\theta_{Enc_{noise}}$ and noise discriminator $\theta_{\mathcal{D}_{noise}}$.
        $Y_{noise}^{fake} = \mathcal{D}_{noise}(\hat{z}_{1:T}), Y_{noise}^{real} = \mathcal{D}_{noise}(z_{1:T}), \quad \hat{z}_{1:T} = Enc_{noise}(h_{1:T}).$
        $\mathcal{L}_{\mathcal{G}}^{noise}(\theta_{Enc_{noise}}) = -\mathbb{E}_{h_{1:T}\sim p(H)}[\mathcal{D}_{noise}(Enc_{noise}(h_{1:T}))].$
        $\mathcal{L}_{\mathcal{D}}^{noise}(\theta_{\mathcal{D}_{noise}}) = \mathbb{E}_{z_{1:T}\sim p(Z)}[\max(0, 1 - \mathcal{D}_{noise}(z_{1:T}))]+$
        $\mathbb{E}_{h_{1:T}\sim p(H)}[\max(0, 1 + \mathcal{D}_{noise}(Enc_{noise}(h_{1:T})))].$
6:  **Latent supervised training.** We compared $h_{1:T}$ and $\hat{h}_{1:T}$ to ensure that $\hat{h}_{1:T}$ from $z_{1:T}$ is similar to the $h_{1:T}$. Compute $\mathcal{L}_S^{latent}$ but optimize the parameters in next stage.
        $\hat{h}_{1:T} = Dec_{noise}(z_{1:T}), \quad h_{1:T} = Dec_{data}(x_{1:T}).$
        $\mathcal{L}_S^{latent}(\theta_{Dec_{noise}}, \theta_{Dec_{data}}, \theta_{\mathcal{D}_{data}}) =$
        $\mathbb{E}_{x_{1:T}\sim p(X), z_{1:T}\sim p(Z)}(\|Enc_{data}(x_{1:T}) - Dec_{noise}(z_{1:T})\|_2).$
7:  **Data discrimination.** We finally combine supervised and unsupervised training to ensure our generated data $\hat{x}_{1:T}$ can match the distribution of $x_{1:T}$ because the generated data is our goal. Compute $\mathcal{L}_{\mathcal{G}}^{data}$ and $\mathcal{L}_{\mathcal{D}}^{data}$ and optimize the generator $(\theta_{Dec_{noise}}, \theta_{Dec_{data}})$ and data discriminator $\theta_{\mathcal{D}_{data}}$.
        $Y_{data}^{fake} = \mathcal{D}_{data}(\hat{x}_{1:T}), Y_{data}^{real} = \mathcal{D}_{data}(x_{1:T}), \quad \hat{x}_{1:T} = Dec_{data}(Dec_{noise}(h_{1:T})).$
        $\mathcal{L}_{\mathcal{G}}^{data}(\theta_{Dec_{noise}}, \theta_{Dec_{data}}) = -\mathbb{E}_{z_{1:T}\sim p(Z)}[\mathcal{D}_{data}(Dec_{data}(Dec_{noise}(z_{1:T})))] + \mathcal{L}_S^{latent}.$
        $\mathcal{L}_{\mathcal{D}}^{data}(\theta_{\mathcal{D}_{data}}) = \mathbb{E}_{x_{1:T}\sim p(X)}[\max(0, 1 - \mathcal{D}_{data}(x_{1:T}))]+$
        $\mathbb{E}_{z_{1:T}\sim p(Z)}[\max(0, 1 + \mathcal{D}_{data}(Dec_{data}(Dec_{noise}(z_{1:T}))))] + \mathcal{L}_S^{latent}.$
8: **end for**
```

5 Experimental Results

Here we will introduce our experimental results. Section 5.1 will introduce which datasets we use. Section 5.2 will introduce our evaluation metric. Section 5.3 will show our visualization results and evaluation results.

5.1 Different Kinds of Datasets

Table 1: Datasets attributes.

Datasets	Instances	The number of features
Stock	3685 (hourly)	6
Demand	60 (daily)	13
Traffic	15917 (hourly)	6
Energy	19735 (minutely)	28

As shown in table 1, to evaluate our model, we choose four different types of time-series datasets for testing. The criteria for selecting these types are divided into whether it is periodic, the number of Instances, and the correlation between time and characteristics. And according to different standard combinations to choose our dataset.

1. **Stock**. This dataset has no periodicity but is continuous in time, and features are related to each other. This data set uses daily stock price data from 2004 to 2019 for google, including trading volume and maximum, minimum, opening, closing, and adjusted closing prices. There are 3685 instances, and the characteristic number of data is 24.

2. **Demand**. It is a cyclical but not completely continuous dataset, and features are related to each other. In addition, the number of data in this dataset is tiny. And this is an actual data set of a large Brazilian logistics company. The data set was collected within 60 days, and the sum of the orders of twelve customers is the total number of orders per day. There are 60 instances, and the characteristic number of data is 13.

3. **Traffic**. It is a dataset with periodicity and continuous time but with less correlation between features. This data set uses the hourly traffic volume of westbound interstate highways from 2012 to 2014, including weather characteristics and traffic volume records. The number of instances is 15,917, and the characteristic number of data items is 6.

4. **Energy**. It is a discrete and irregular data set. The data includes the power consumption and temperature and humidity records of each room. The number of instances is 19,735, and the number of data features is 28.

5.2 Evaluation Metrics

Here we introduce our evaluation method. We use the visualization results to evaluate whether the distribution of our synthetic data is similar to that of the actual data. Then, by establishing a simple classification model and prediction model, we evaluate the specific similarity of the data and the extent to which the judgment-generated data retains the temporal characteristics of the original data.

Visualization We use PCA [34] and t-SNE [35] to visualize the real data and the generated data in two dimensions. The visualization results are used to determine whether the distributions of real data and synthetic data are similar. If they are all in the same interval and the distribution has a high degree of similarity, the synthetic data we generate is identical to the real data. Among them, t-SNE is a nonlinear dimensionality reduction method. The algorithm is random. Multiple experiments can produce different results, but it preserves the relationship between the data and its neighbors. Generally, PCA is deterministic, and the result after each calculation is the same.

Classification model During the test, we trained a simple (two-layer GRU) time series classification model to distinguish real data from synthetic data, evaluating the similarity between the two data.

Classified score Assuming that the classifier cannot find the difference between the synthetic data and the real data, the accuracy of the classifier should be 50%, so we set the classification score to $\|\text{classificationaccuracy} - 0.5\|$. The lower the score we get, the more similar the synthetic data.

Prediction model We trained a simple (5-layer GRU model) time series prediction model to predict the next step of each input sequence during the test.

Predictive score To verify whether the time characteristics of the synthetic data are similar to the original data. The evaluation score is based on mean absolute error (MAE). The lower the score, the more similar the temporal characteristics we get.

We train both the classification score and the prediction score ten times and get their average score. At the same time, we divide the predictive score into TRTS (training on real data, testing on synthetic data), TSTR (training on synthetic data, testing on real data), TRTR (do not perform any data amplification, only train on real data, and test on real data). The ratio of our training set to test set is 7 : 3.

5.3 Results

Section 5.3.1 will introduce the experimental results and comparisons of time-series data augmentation methods, including our method on different data sets. In section 5.3.2, we will present the results of using other loss functions in our approach. In section 5.3.3, we will test the improvement of the prediction model by adding synthetic data for training.

5.3.1 Comparison of Different Datasets

Here, we will provide C-RNN-GAN [13] and TimeGAN [1] as comparison objects. We use C-RNN-GAN, TimeGAN, and our method to visualize four datasets and compare the distribution of synthetic data generated by different methods. Use a classified score to compare the similarity between the data generated by each method and the original data. Use a predictive score to compare the completeness of the time characteristics of the data retention caused by each method. First, let's look at the results on Stock.

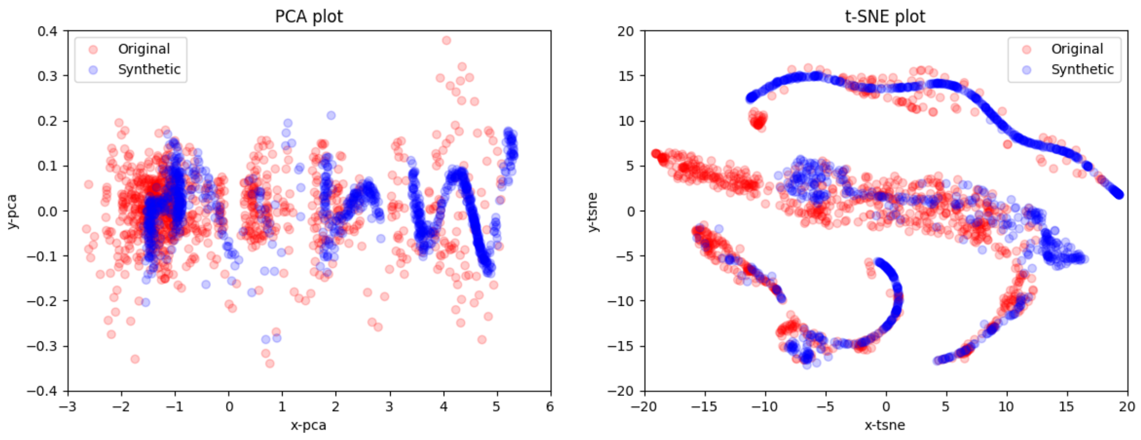


Figure 8: Visualization of stock dataset using C-RNN-GAN.
The figure from left to right shows the PCA and t-SNE visualization of C-RNN-GAN on the stock dataset. The blue dots represent the distribution of our synthetic data in a two-dimensional space, and the red dots represent real data distribution.

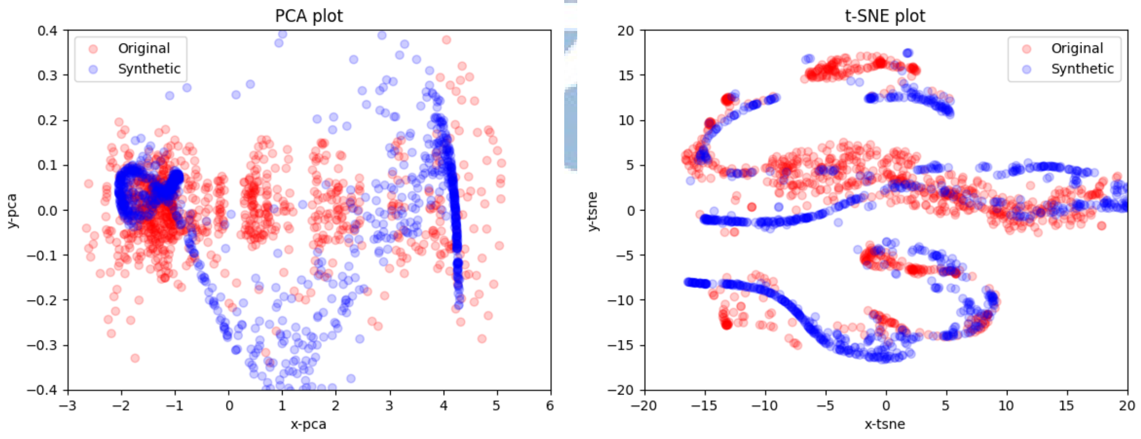


Figure 9: Visualization of stock dataset using TimeGAN.
The figure from left to right shows the PCA and t-SNE visualization results of TimeGAN on the stock dataset. The blue dots represent the distribution of our synthetic data in a two-dimensional space, and the red dots represent real data distribution.

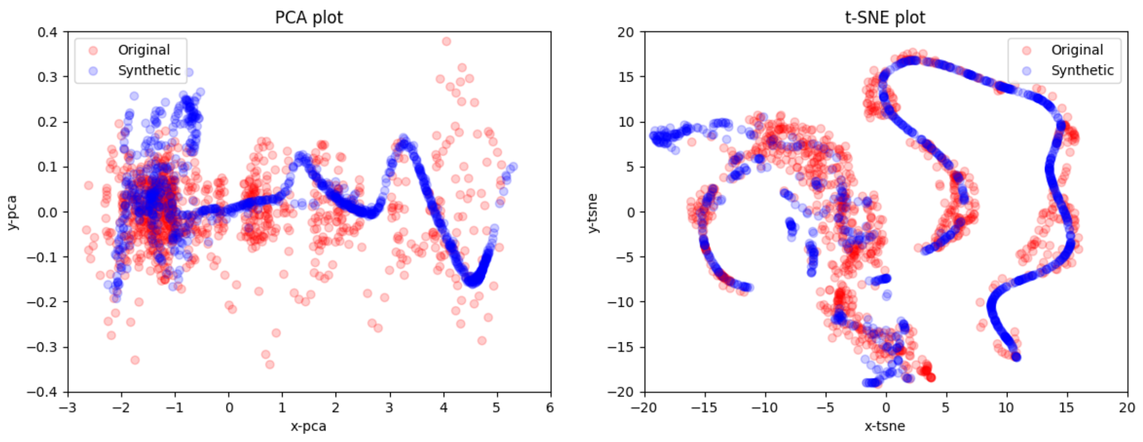


Figure 10: Visualization of stock dataset using our method.
The figure from left to right shows the PCA and t-SNE visualization results of our method on the stock dataset. The blue dots represent the distribution of our synthetic data in a two-dimensional space, and the red dots represent real data distribution.

From the figure 8, 9, 10, the PCA results show that CRNNGAN and our technique has excellent overlap with real data, and the similarity is also high. But there is almost no overlap in the middle of TimeGAN. The result of t-SNE shows that some of the blue dots in C-RNN-GAN did not cover the red dots, while some of the blue dots in TimeGAN ran out of the real data range. Our method is to cover some of the red dots sparsely. Next, look at the classified score and predictive score of each method on the Stock dataset.

Table 2: Stock results.

Stock dataset	Score			
	Classified score↓	Predictive↓		
		TRTS	TSTR	TRTR
C-RNN-GAN	0.2014	0.3969	0.3746	
TimeGAN	0.3984	0.9560	0.7607	0.1124
Ours	**0.0929**	**0.1996**	**0.1229**	

Among them, the TRTS result verifies whether some synthetic data is not within the scope of the real data distribution. If the synthetic data goes to the area of non-real data, the score will be higher. The TSTR result verifies whether some real data is out of the range of synthetic data distribution. The score will also be higher if the real data range does not wholly cover the synthetic data. TRTR is the benchmark used for evaluation without any synthetic data. From table 2 We can see that our method has the highest similarity and can also learn the most complete time characteristics.

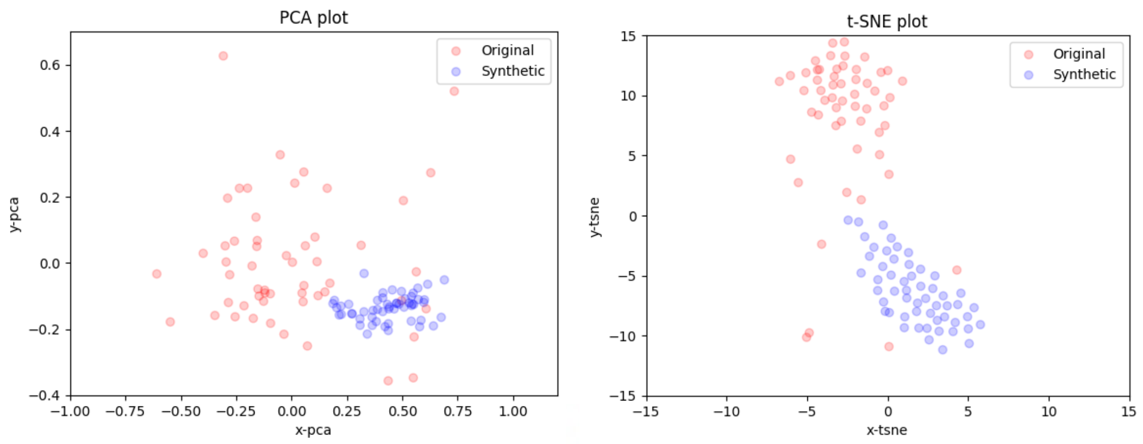


Figure 11: Viusalization of demand dataset using C-RNN-GAN.
The figure from left to right shows the PCA and t-SNE visualization of C-RNN-GAN on the demand dataset. The blue dots represent the distribution of our synthetic data in a two-dimensional space, and the red dots represent real data distribution.

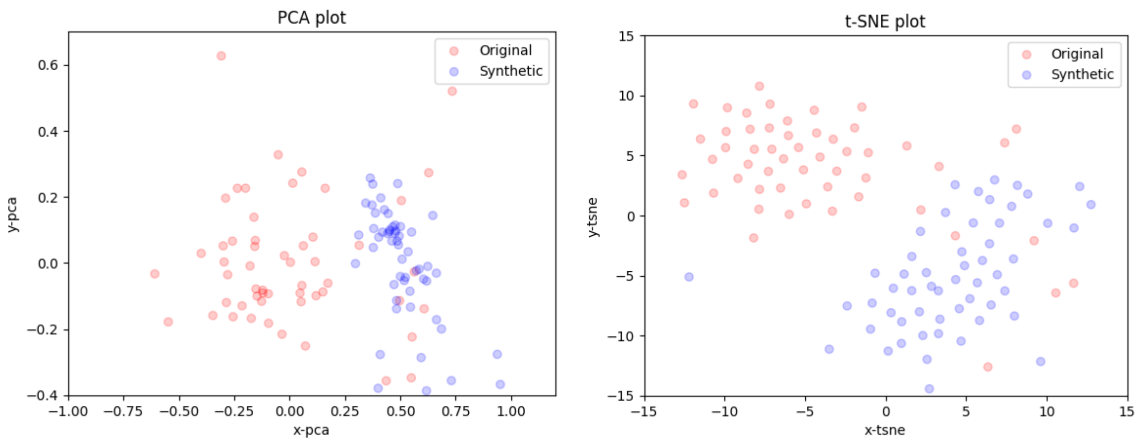


Figure 12: Viusalization of demand dataset using TimeGAN.

The figure from left to right shows the PCA and t-SNE visualization results of TimeGAN on the demand dataset. The blue dots represent the distribution of our synthetic data in a two-dimensional space, and the red dots represent real data distribution.

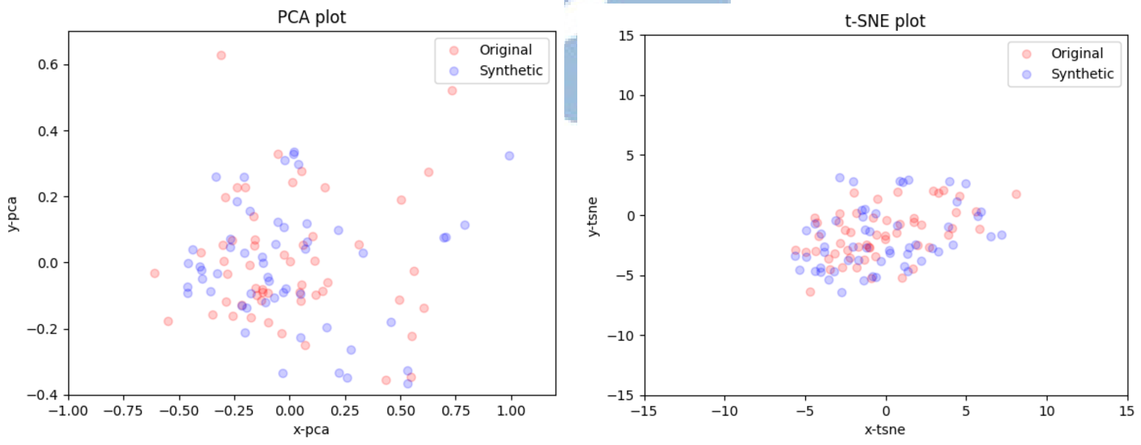


Figure 13: Viusalization of demand dataset using our method.

The figure from left to right shows the PCA and t-SNE visualization results of our method on the demand dataset. The blue dots represent the distribution of our synthetic data in a two-dimensional space, and the red dots represent real data distribution.

From the figure 11, 12, 13, the results of PCA can illustrate our synthetic The data are all in the distribution of real data and are very similar, and fill the holes in the real distribution. The distributions of the first two results are not in the same interval, and the distributions are not similar. Next, look at the classified score and predictive score of each method on the demand dataset.

Table 3: Demand results.

Demand dataset	Score			
	Classified score↓	Predictive↓		
		TRTS	TSTR	TRTR
C-RNN-GAN	0.2912	0.5118	0.2867	0.2663
TimeGAN	0.1765	0.2826	0.2933	
Ours	**0.1588**	**0.2749**	**0.2628**	

From the classified score results of table 3, we can see that the data generated by our method has the highest similarity to the original data, and TSTR and TRTS show that our method can learn the complete time characteristics. Next we look at the traffic results.

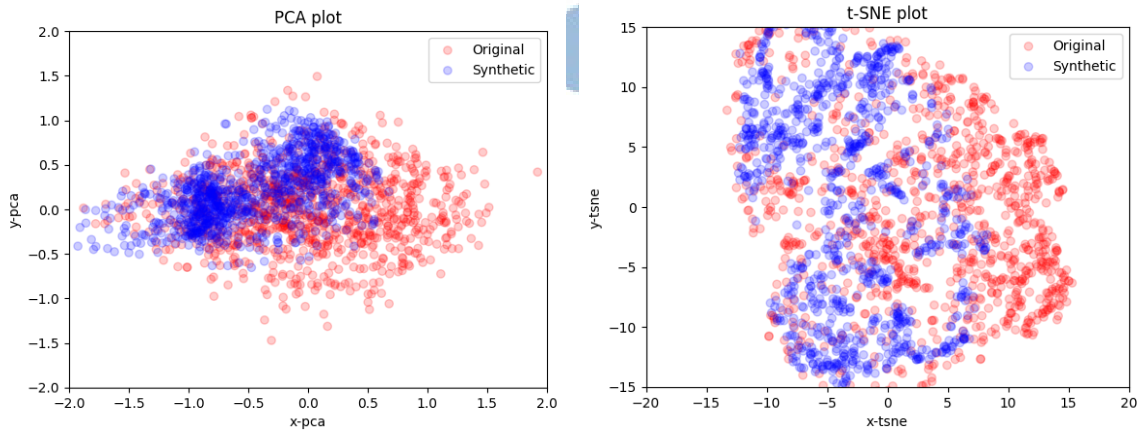


Figure 14: Visualization of traffic dataset using C-RNN-GAN.
The figure from left to right shows the PCA and t-SNE visualization of C-RNN-GAN on the traffic dataset. The blue dots represent the distribution of our synthetic data in a two-dimensional space, and the red dots represent real data distribution.

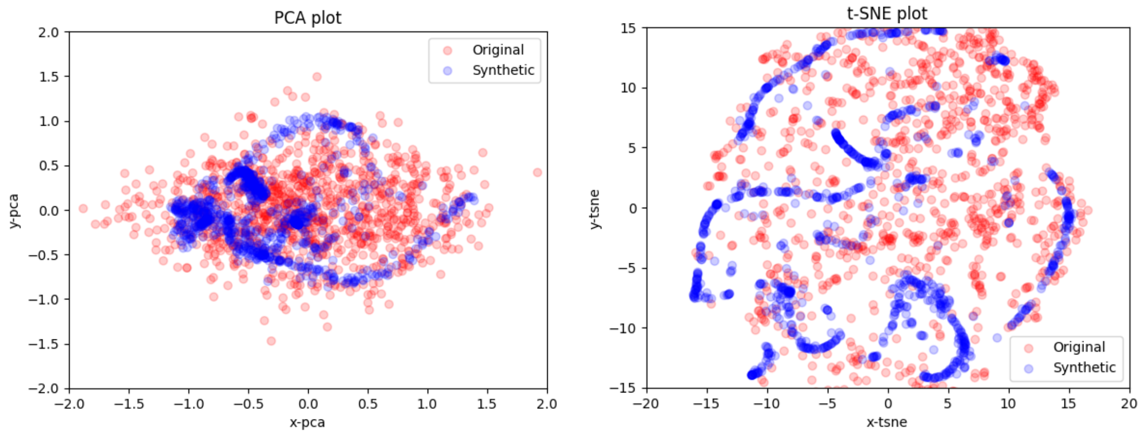


Figure 15: Viusalization of traffic dataset using TimeGAN.
The figure from left to right shows the PCA and t-SNE visualization results of TimeGAN on the traffic dataset. The blue dots represent the distribution of our synthetic data in a two-dimensional space, and the red dots represent real data distribution.

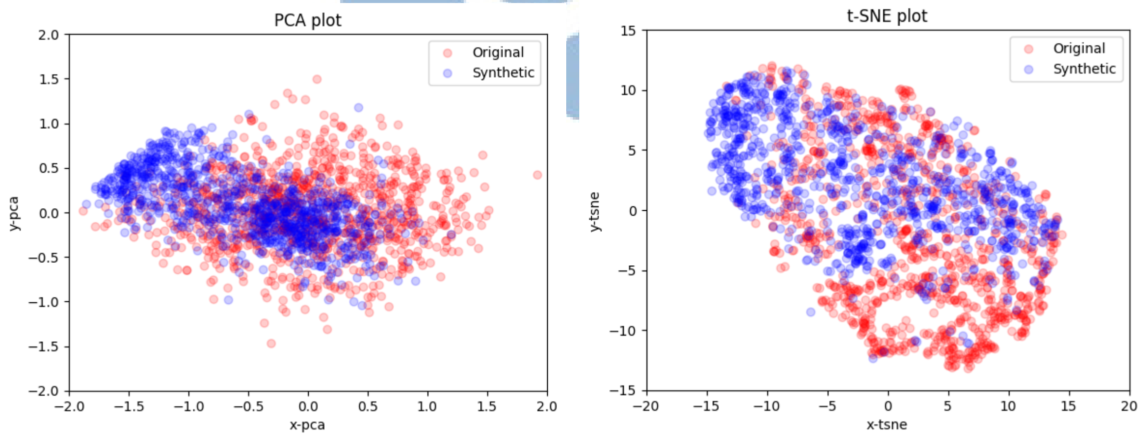


Figure 16: Viusalization of traffic dataset using Our method.
The figure from left to right shows the PCA and t-SNE visualization results of our method on the traffic dataset. The blue dots represent the distribution of our synthetic data in a two-dimensional space, and the red dots represent real data distribution.

From the figure 14, 15, 16, the blue dots can be seen from the results of CRNNGAN The red dots are not entirely covered, and a small part of the blue dots of the PCA is not within the red dot range. From the results of TimeGAN, we can see that the blue dots are primarily concentrated in a specific block, and the similarity is not high. From our results, we can see that the blue dots are not entirely covered on the red dots, but there is still a tiny part of the blue dots distributed on the uncovered area, but a small amount of the blue dots in PCA are not within the range of the red dot.

Table 4: Traffic results.

Traffic dataset	Score			
	Classified score↓	Predictive↓		
		TRTS	TSTR	TRTR
C-RNN-GAN	0.4963	**0.3483**	0.5643	0.4814
TimeGAN	0.4980	0.3754	0.5504	
Ours	**0.1469**	0.4672	**0.5152**	

Judging from the classified score results of table 4, our method has the highest similarity. According to TRTS data, CRNNGAN has the slightest blue dots, while our method has more deviations from the PCA in the upper left corner of the blue dots. The data from TSTR shows that the blue point of our method has the highest coverage on the red point.

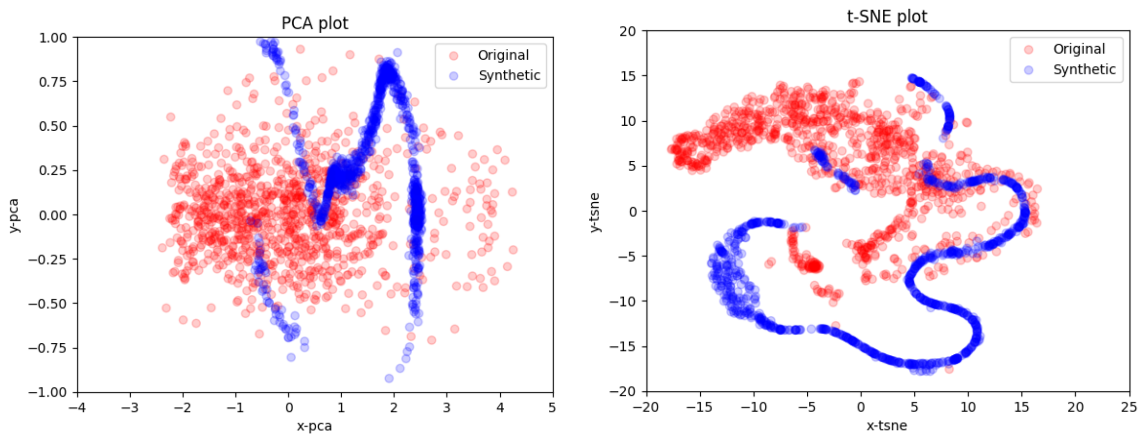


Figure 17: Viusalization of energy dataset using C-RNN-GAN.
The figure from left to right shows the PCA and t-SNE visualization of C-RNN-GAN on the energy dataset. The blue dots represent the distribution of our synthetic data in a two-dimensional space, and the red dots represent real data distribution.

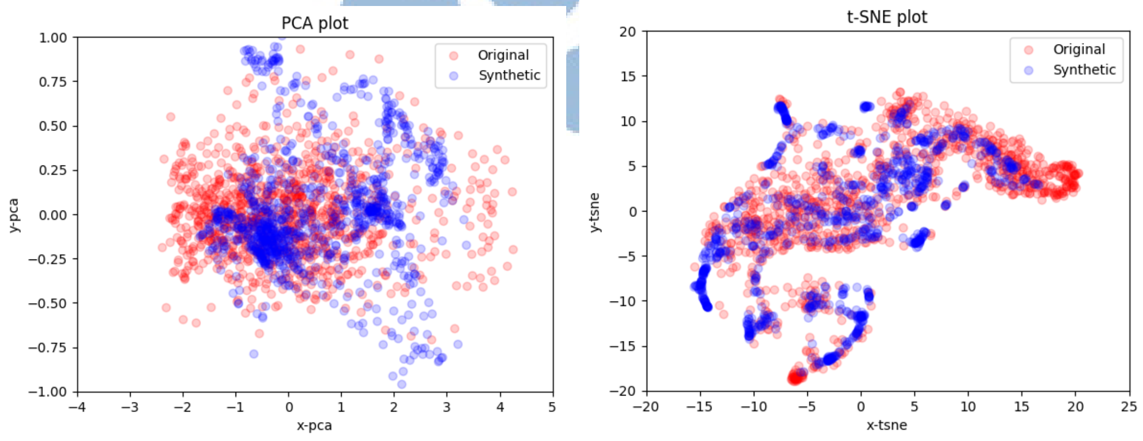


Figure 18: Viusalization of energy dataset using TimeGAN.
The figure from left to right shows the PCA and t-SNE visualization of TimeGAN on the energy dataset. The blue dots represent the distribution of our synthetic data in a two-dimensional space, and the red dots represent real data distribution.

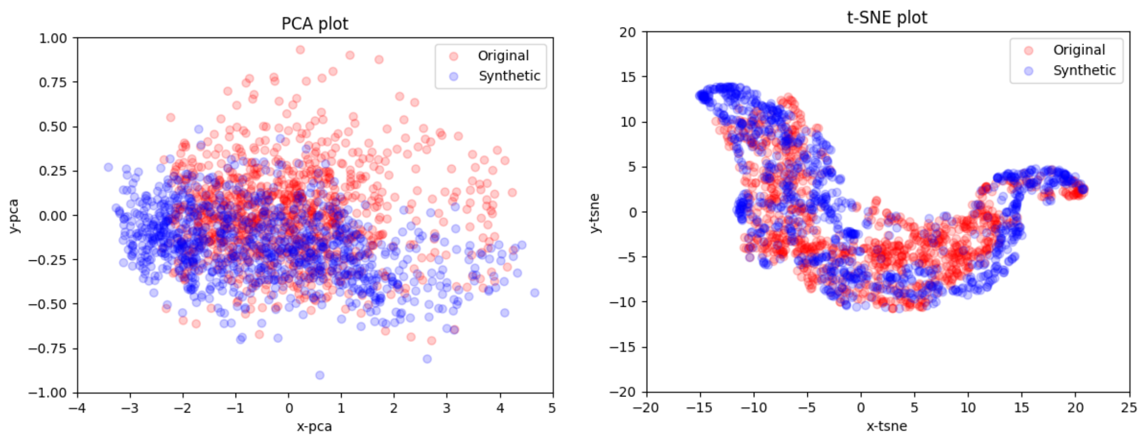


Figure 19: Viusalization of energy dataset using Our method.

The figure from left to right shows the PCA and t-SNE visualization results of our method on the energy dataset. The blue dots represent the distribution of our synthetic data in a two-dimensional space, and the red dots represent real data distribution.

From the figure 17, 18, 19, we can see the similarity of the distribution is not high from the results of C-RNN-GAN, and the blue dots are concentrated on some blocks and deviate from the red dot range. From the PCA results of TimeGAN, it can be seen that many blue dots deviate from the red dot area, but the t-SNE results are primarily concentrated in the distribution of real data. Our results show that the blue dots are not entirely covered on the red dots, but a tiny part of the blue dots is still sparsely distributed on the uncovered area.

Table 5: Energy results.

Energy dataset	Score			
	Classified score↓	Predictive↓		
		TRTS	TSTR	TRTR
C-RNN-GAN	0.4993	**0.1996**	0.6334	
TimeGAN	0.4985	0.2674	**0.5651**	0.4841
Ours	**0.3678**	0.4624	0.5716	

Judging from the classified score results of table 5, our method has the highest similarity. Among them, the TRTS result of C-RNN-GAN is the best, the TSTR result of TimeGAN is the best.

5.3.2 Different Loss Function of Our Methods

In [26, 27], it mentions the problems of GAN [12], including the issue of gradient disappearance caused by the imbalance of discriminator and generator during training and the problem of mode collapse of a generator. The issues mentioned above are caused by the imperfect design of the original GAN loss function. In WGAN [26], the problems of GAN are discussed, and a new loss function is proposed to solve the above problems, which almost solves the instability of GAN training. WGAN-GP [27] has improved the loss function of WGAN [26] to achieve better results. Recently, Geometric GAN [25] geometrically reinterpreted the GAN architecture, and based on this, proposed a new loss function hinge loss, which is the method used in this article. To explore which loss function is most helpful to the method proposed in this article, we use different loss functions for four datasets to test which scheme works best.

Table 6: Different loss functions on stock results.

Stock dataset $\mathcal{L}_{\mathcal{G}}^{noise}, \mathcal{L}_{\mathcal{D}}^{noise}, \mathcal{L}_{\mathcal{G}}^{data}, \mathcal{L}_{\mathcal{D}}^{data}$	Score			
	Classified score↓	Predictive↓		
		TRTS	TSTR	TRTR
Binary cross entropy with logits	0.1125	**0.1070**	**0.1149**	0.1124
Hinge loss (Ours)	**0.0929**	0.1996	0.1229	
WGAN-GP	0.3017	0.1112	0.1111	

Table 7: Different loss functions on demand results.

Demand dataset $\mathcal{L}_{\mathcal{G}}^{noise}, \mathcal{L}_{\mathcal{D}}^{noise}, \mathcal{L}_{\mathcal{G}}^{data}, \mathcal{L}_{\mathcal{D}}^{data}$	Score			
	Classified score↓	Predictive↓		
		TRTS	TSTR	TRTR
Binary cross entropy with logits	0.2000	**0.2517**	0.3204	0.2663
Hinge loss (Ours)	0.1588	0.2749	**0.2628**	
WGAN-GP	**0.1500**	0.4789	0.4241	

Table 8: Different loss functions on traffic results.

Traffic dataset $\mathcal{L}_{\mathcal{G}}^{noise}, \mathcal{L}_{\mathcal{D}}^{noise}, \mathcal{L}_{\mathcal{G}}^{data}, \mathcal{L}_{\mathcal{D}}^{data}$	Score			
	Classified score↓	Predictive↓		
		TRTS	TSTR	TRTR
Binary cross entropy with logits	0.3094	0.5093	0.5890	
Hinge loss (Ours)	**0.1469**	**0.4672**	**0.5152**	0.4814
WGAN-GP	0.2650	0.4781	0.5438	

Table 9: Different loss functions on energy results.

Energy dataset $\mathcal{L}_{\mathcal{G}}^{noise}, \mathcal{L}_{\mathcal{D}}^{noise}, \mathcal{L}_{\mathcal{G}}^{data}, \mathcal{L}_{\mathcal{D}}^{data}$	Score			
	Classified score↓	Predictive↓		
		TRTS	TSTR	TRTR
Binary cross entropy with logits	0.4367	**0.4658**	0.6473	
Hinge loss (Ours)	**0.3678**	0.4924	**0.5716**	0.4841
WGAN-GP	0.4000	0.5030	0.5958	

Judging from table 6, 7, 8, 9, the three loss functions have their advantages. From the simpler stock and demand datasets, Binary cross-entropy with logits, which is the original loss function proposed by GAN, gets the best results, followed by hinge loss and wgan-gp last. But in terms of traffic and energy datasets, Binary cross-entropy with logits has the worst effect. Hinge loss has the best performance, followed by WGAN-GP. From the above results, we can see that Binary cross-entropy with logits has advantages on simpler datasets. On more complex datasets, hinge loss, wgan-gp has more stable performance.

5.3.3 Improvement of the Prediction Model

In this chapter, we will conduct mixed training on real data and synthetic data and test on real data, hereinafter referred to as TMTR, to test whether the synthetic data we generate can improve the performance of the prediction model. We will then use stock and demand datasets to experiment and attach the results of other methods. If TMTR is lower than TRTR, it means that the data we generate can ultimately improve the performance of the prediction model.

Table 10: Stock results with TMTR.

Stock dataset	Score				
	Classified score↓	Predictive↓			
		TRTS	TSTR	TMTR	TRTR
C-RNN-GAN	0.2014	0.3969	0.3746	0.1590	
TimeGAN	0.3984	0.9560	0.7607	0.2540	0.1124
Ours	**0.0929**	**0.1996**	**0.1229**	**0.0931**	

Table 11: Demand results with TMTR.

Demand dataset	Score				
	Classified score↓	Predictive↓			
		TRTS	TSTR	TMTR	TRTR
C-RNN-GAN	0.2912	0.5118	0.2867	0.2729	0.2663
TimeGAN	0.1765	0.2826	0.2933	0.2637	
Ours	**0.1588**	**0.2749**	**0.2628**	**0.2532**	

From table 10, 11, we can find that the TMTR of our method is lower than TRTR, and the improvement of our method to the prediction model in the three ways is the highest.

Finally, we can find that the closer the results of TSTR and TRTS are, the better the results of TMTR. Therefore, we can conclude that the closer the results of TSTR and TRTS are, the more complete the time characteristics of the original data can be learned. The experimental results show that our method can learn the most accurate data distribution and whole time characteristics compared with the other two methods.

6 Conclusion

In this article, we propose a TimeGAN-based time series data augmentation model to generate a variety of time series data. By mapping data to different domains to strengthen the connection between multiple domains, the model can learn the characteristics and distribution of data more thoroughly. Experimental results show that our method can generate data with the complete time characteristics to help improve the performance of the model, and our method can learn the complete data distribution. In the future, we hope to join the transformer architecture to accelerate training and strengthen the ability to learn complex features to handle more types of time series data.

References

[1] J. Yoon, D. Jarrett, and M. van der Schaar, "Time-series generative adversarial networks," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019. i, 1, 2, 5, 6, 7, 18

[2] Z. Han, J. Zhao, H. Leung, K. F. Ma, and W. Wang, "A review of deep learning models for time series prediction," *IEEE Sensors Journal*, vol. 21, no. 6, pp. 7833–7848, 2019.

[3] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey," *arXiv preprint arXiv:1901.03407*, 2019. 1

[4] A. Le Guennec, S. Malinowski, and R. Tavenard, "Data augmentation for time series classification using convolutional neural networks," in *ECML/PKDD workshop on advanced analytics and learning on temporal data*, 2016. 1, 3

[5] Z. Cui, W. Chen, and Y. Chen, "Multi-scale convolutional neural networks for time series classification," *arXiv preprint arXiv:1603.06995*, 2016. 1, 3

[6] T. Wen and R. Keyes, "Time series anomaly detection using convolutional neural networks and transfer learning," *arXiv preprint arXiv:1905.13628*, 2019. 1, 3

[7] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, "Data augmentation using synthetic data for time series classification with deep residual networks," *arXiv preprint arXiv:1808.02455*, 2018. 1, 3

[8] J. Gao, X. Song, Q. Wen, P. Wang, L. Sun, and H. Xu, "Robusttad: Robust time series anomaly detection via decomposition and convolutional neural networks," *arXiv preprint arXiv:2002.09545*, 2020. 1, 3

[9] T. E. K. Lee, Y. Kuah, K.-H. Leo, S. Sanei, E. Chew, and L. Zhao, "Surrogate rehabilitative time series data for image-based deep learning," in *2019 27th European Signal Processing Conference (EUSIPCO)*, pp. 1–5, IEEE, 2019. 1, 3

[10] T. DeVries and G. W. Taylor, "Dataset augmentation in feature space," *arXiv preprint arXiv:1702.05538*, 2017. 1, 4

[11] Q. Wen, L. Sun, F. Yang, X. Song, J. Gao, X. Wang, and H. Xu, "Time series data augmentation for deep learning: A survey," *arXiv preprint arXiv:2002.12478*, 2020. 1

[12] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014. 1, 4, 28

[13] O. Mogren, "C-rnn-gan: A continuous recurrent neural network with adversarial training," in *Constructive Machine Learning Workshop (CML) at NIPS 2016*, p. 1, 2016. 1, 4, 18

[14] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016. 1, 4

[15] C. Donahue, J. McAuley, and M. Puckette, "Adversarial audio synthesis," in *ICLR*, 2019. 1, 4

[16] P. Senin, "Dynamic time warping algorithm review," *Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA*, vol. 855, no. 1-23, p. 40, 2008. 3

[17] M. Cuturi and M. Blondel, "Soft-dtw: a differentiable loss function for time-series," in *International Conference on Machine Learning*, pp. 894–903, PMLR, 2017. 3

[18] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006. 4

[19] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013. 4

[20] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv preprint arXiv:1803.01271*, 2018. 4

[21] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther, "Autoencoding beyond pixels using a learned similarity metric," in *International conference on machine learning*, pp. 1558–1566, PMLR, 2016. 4

[22] J. Donahue, P. Krähenbühl, and T. Darrell, “Adversarial feature learning,” *arXiv preprint arXiv:1605.09782*, 2016. 4

[23] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, “Adversarial autoencoders,” *arXiv preprint arXiv:1511.05644*, 2015. 4

[24] C. Esteban, S. L. Hyland, and G. Rätsch, “Real-valued (medical) time series generation with recurrent conditional gans,” *arXiv preprint arXiv:1706.02633*, 2017. 4

[25] J. H. Lim and J. C. Ye, “Geometric gan,” *arXiv preprint arXiv:1705.02894*, 2017. 4, 28

[26] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein generative adversarial networks,” in *International conference on machine learning*, pp. 214–223, PMLR, 2017. 4, 28

[27] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, “Improved training of wasserstein gans,” *arXiv preprint arXiv:1704.00028*, 2017. 4, 28

[28] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, “Spectral normalization for generative adversarial networks,” *arXiv preprint arXiv:1802.05957*, 2018. 4

[29] A. Radford, L. Metz, and S. Chintala, “Deep convolutional generative adversarial network,” in *Under review as a conference paper at ICLR*, 2016. 4

[30] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, “Self-attention generative adversarial networks,” in *International conference on machine learning*, pp. 7354–7363, PMLR, 2019. 4

[31] A. Brock, J. Donahue, and K. Simonyan, “Large scale gan training for high fidelity natural image synthesis,” *arXiv preprint arXiv:1809.11096*, 2018. 4

[32] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” *arXiv preprint arXiv:1406.1078*, 2014. 14

[33] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014. 14

[34] F. B. Bryant and P. R. Yarnold, "Principal-components analysis and exploratory and confirmatory factor analysis.," 1995. 17

[35] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne.," *Journal of machine learning research*, vol. 9, no. 11, 2008. 17