Stock Market Missing Data Imputation

Yue Hu, January 2023

https://github.com/YueHu777/Missing-Data-MLCOE

1 Abstract

Time series missing data imputing has long been a challenging topic in machine learning. Recently, the CSDI and SSSD model have shown state of the art results on long sequence missing data imputing. In this report, we first implemented these two models in Tensorflow. We then applied them to the financial stock market and compared the performance of the models. As a result, we found that the CSDI model outperforms SSSD in this case and conducted a brief discussion on why SSSD is not suitable for financial data.

2 Introduction

Missing data is a common phenomenon in real-world machine learning applications, and time series data is particularly prevalent in this case. Since most algorithms require data without missing values to train, imputing missing input data is a major challenge for machine learning applications.

In the stock market, due to different holidays in the USA, Europe and Hong Kong market, there are missing data in some of the dates. In order to impute the data, we implemented the SSSD model which has shown state-of-the-art results on a lot of datasets in Tensorflow. We also implemented the CSDI model as a baseline model for comparison. Both of the models enjoy a similarly DiffWave-like[9] model architecture but the main difference between them is the way they capture long range dependencies. The SSSD model uses a S4 layer[5] while the CSDI model uses a transformer layer[13]. The key idea of the S4 model is to formalize the temporal dependency by a state space model with specific state matrices. And the transformer relies entirely on an attention mechanism to draw global dependencies between input and output.

The remainder of this report is organized as follows. Section 3 is a literature review on the related works. Section 4 briefly introduces the methodologies used in the models. Section 5 is the answer to the specific interview questions, where section 5.2.2 shows the assumptions for the model to apply, 5.2.3 shows a detailed data pre-processing procedure, and 5.2.5 compares the results of the two models and demonstrates that CSDI outperforms SSSD regarding the financial data. In section 6 we discuss the reason why SSSD does not work is probably because that the HiPPO matrix, key of the S4 layer, may not be suitable to be applied to the financial data which is a brownian motion. Finally, we conclude this report in section 7.

3 Literature Review

There are numerous of existing studies focusing on imputing time series missing values. One of the most popular techniques is to use the generative models such as GANs[2], VAE[8] and diffusion models[12] as imputers, and combine them with some sequence model to capture temporal dependency[3].

However, how to efficiently model sequential data for long time steps is still a main challenge in machine learning. Recurrent neural networks (RNNs), temporal convolutions, and neural differential equations (NDEs) are popular families of deep learning models for time-series data, each with unique strengths and trade-offs in modeling power and computational efficiency[6]. For example, RNNs are natural stateful models for sequential data that require only constant computation/storage per time step, but are slow to train and suffer from optimization difficulties (e.g., the "vanishing gradient problem"[11]). CNNs encode local context and enjoy fast and parallelizable training, but are not sequential. NDEs are principled mathematical models that can theoretically address continuous-time problems and long-term dependencies [10], but are very inefficient.

Besides these models, the Transformer model relies entirely on an attention mechanism to draw global dependencies between input and output and has acheived great results in sequence modeling[14]. And recently, Gu et al. [6] presented an innovative approach for long range sequence using structured state space sequence (S4) layers.

The S4 layer can be viewed as a generalization of CNNs and RNNs and inherit their strengths[6]. It is inspired by control systems and models sequences by simulating the fundamental state space model (SSM) $x_k = \bar{A}x_{k-1} + \bar{B}u_k, y_k = \bar{C}x_k + \bar{D}u_k$. This formulation can also be written as a discrete convolution $y = \bar{K} * u, \bar{K} = (\bar{C}\bar{B}, \bar{C}\bar{A}\bar{B}, ..., \bar{C}\bar{A}^{L-1}\bar{B})$. With a particular initialization of matrix A according to HiPPO theory[4], it can capture the long-term dependencies well. Given the convolution representation $\bar{K}$, in order to compute it efficiently, Gu et al decomposed the HiPPO matrix as the sum of a normal and low-rank matrix and then diagonalized it. Fast Fourier transforms (FFTs) can then be applied afterwards to efficiently parallelize the computation.

Based on the generative models and sequence models mentioned above, Tashiro et al constructed the Conditional Score-based Diffusion Models (CSDI) [13] and Alcaraz et al proposed the structured state space diffusion (SSSD) models [1] for time series missing data imputation. Both of the two models used the DiffWave structure (a ResNet) as the diffusion model, and the main difference between the two models is that CSDI used Tranformers as diffusion layers within each of its residual blocks while SSSD used the S4 layers as the diffusion layer to capture long-range dependencies.

According to [1], the SSSD model has outperformed CSDI on various of datasets including audio, image and healthcare data. In this report, we evaluate both models on the financial time series data and assess the performance of the two models.

4 Methodology Representation

4.1 Diffusion Model

Diffusion models represent a class of generative models that demonstrated promising performance on a range of different data modalities. It learn a mapping from latent space to signal space by sequentially learning to remove noise in a backward process that was added sequentially in a Markovian fashion during a so called forward (diffusion) process. These two processes, therefore, represent the backbone of the diffusion model[1]. A figure illustration is shown in Figure 1.

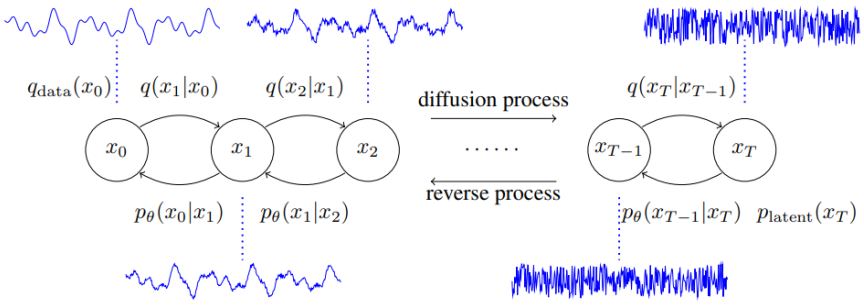

Figure 1: The forward and reverse process in diffusion probabilistic models. [9]

The forward process is parameterized as

$$q(x_1, x_2, ..., x_T|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$$

where $q(x_t|x_{t-1}) = N(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$ and β_t adjusts the noise level. Equivalently, x_t can be expressed in a closed form as $X_t = \sqrt{\alpha_t}x_0 + \sqrt{1-\alpha_t}\epsilon$ for $\epsilon \sim N(0,1)$ where $\alpha_t = \sum_{i=1}^{t}(1-\beta_t)$. The backward process is parameterized as

$$p_\theta(x_0, x_1, ..., x_{T-1}|x_T) = \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$$

where $x_T \sim N(0,1)$. $p_\theta(x_{t-1}|x_t)$ is assumed as normal distributed with learnable parameters. Using a particular parameterization of $p_\theta(x_{t-1}|x_t)$, Ho et al.[7] showed that the reverse process can be trained using the following objective,

$$L = min_\theta E_{x_0 \sim D, \epsilon \sim N(0,1), t \sim U(1,T)} ||\epsilon - \epsilon_\theta(\sqrt{\alpha_t}x_0 + \sqrt{1-\alpha_t}\epsilon, t)||_2^2$$

where D refers to the data distribution and ϵ_θ is parameterized using a neural network. In this report, we use diffusion model for generative model to impute missing data.

4.2 Structured State Space Model

The state space model represent a modeling paradigm to capture in particular long-term dependencies in time series. At its heart, the formulation shown below draws a linear state space transition equation, connecting a one-dimensional input sequence $u(t)$ to a one-dimensional output sequence $y(t)$ via a N-dimensional hidden state $x(t)$ [6].

$$x^{'}(t) = Ax(t) + Bu(t)$$

$$y(t) = Cx(t) + Du(t)$$

With a particular initialization of matrix A according to HiPPO theory, it can capture the long-term dependencies well.

In order to derive the HiPPO matrix, Gu[4] first approximated the time series $f_{\leq t} := f(x)|_{x \leq t}$ with some polynomial $g^{(t)}$ where

(1) the approximation minimizes the error:$||f_{\leq t} - g^{(t)}||_{L2(\mu^{(t)})}$, and

(2) The polynomial $g^{(t)}$ can be mapped to the coefficients c(t) of the basis of orthogonal polynomials defined with respect to the measure $\mu^{(t)}$.

Here $\mu^{(t)}$ can be translated Legendre (LegT), translated Laguerre (LagT) or scaled Legendre measure (LegS), which are different types of orthogonal polynomials with different derivative properties. The HiPPO matrix can then be derived based on these properties.

Later, with some discretization methods, the state space model can be viewed as a generalization of CNN and be computed efficiently.

$$x_k = \bar{A}x_{k-1} + \bar{B}u_k, y_k = \bar{C}x_k + \bar{D}u_k$$

$$y = \bar{K} * u, \bar{K} = (\bar{C}\bar{B}, \bar{C}\bar{A}\bar{B}, ..., \bar{C}\bar{A}^{L-1}\bar{B})$$

The three version of the linear state space layer is shown in Figure 2.

In the SSSD model, it put forwards a Structured State Space Sequence model (S4) based on the SSSM structure to capture temporal dependency.

4.3 Transformer

4.3.1 Self-Attention Mechanism

The fundamental of the Transformer is the self-attention mechanism. An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key[14]. See Figure 3. In practice, the model computes the attention function on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V. The matrix of outputs is:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

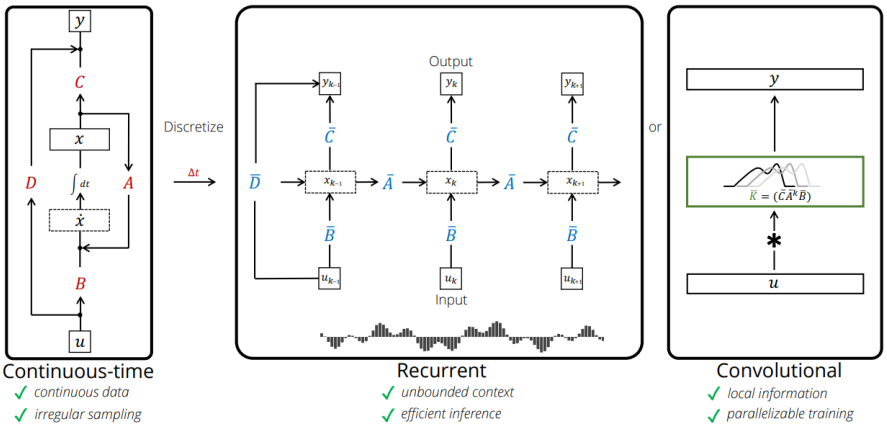

Figure 2: Three views of the linear state space layer[6].(Left) As an implicit continuous model, $x^{'}(t) = Ax(t) + Bu(t), y(t) = Cx(t) + Du(t)$. (Center) As a recurrent model after discretization, $x_k = \bar{A}x_{k-1} + \bar{B}u_k, y_k = \bar{C}x_k + \bar{D}u_k$. (Right) As a convolutional model, $y = \bar{K} * u, \bar{K} = (\bar{C}\bar{B}, \bar{C}\bar{A}\bar{B}, ..., \bar{C}\bar{A}^{L-1}\bar{B})$

As $softmax(\frac{QK^T}{\sqrt{d_k}})$ models pairwise dependency weights, it is used to deliver a flexible and powerful mechanism to learn long-distance dependencies in the CSDI model.

4.3.2 Position Embedding

In addition, since the model is based on self-attention and contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, information about the relative or absolute position of the tokens in the sequence must be injected. To this end, a sinusoidal version of "positional encodings" is added to the input embeddings. The sine and cosine functions of different frequencies are as follow:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}}$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}}$$

where pos is the position, i is the dimension and d_{model} is the model embedding dimension. This function allows the model to easily learn to attend by relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}.

4.4 CSDI and SSSD

Based on the methodologies mentioned above, Tashiro et al constructed the Conditional Score-based Diffusion Models (CSDI) [13] and Alcaraz et al proposed the structured state space diffusion (SSSD) models [1] for time series missing data imputation. Both of the two models used the DiffWave[9] architecture,which is composed of multiple residual layers, as the conditional denoising function ϵ_θ,

[1]Source: https://zhuanlan.zhihu.com/p/47282410

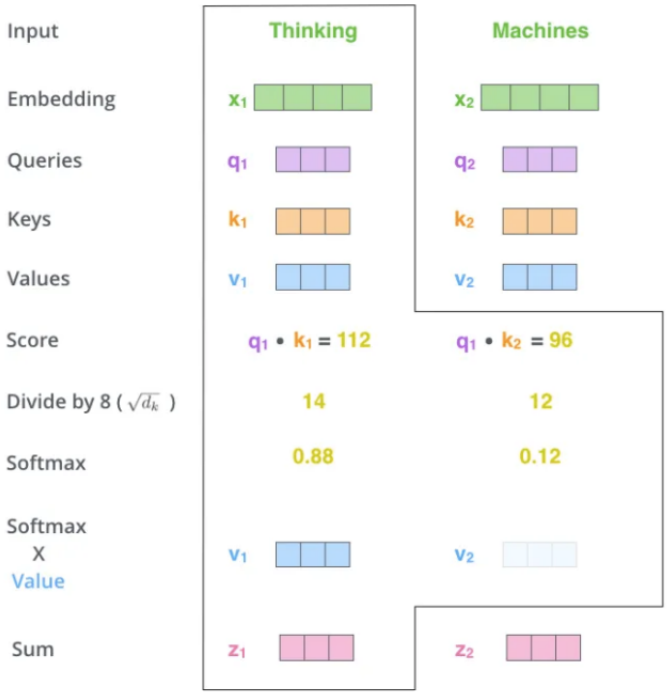

Figure 3: The self-attention mechanism[1]

and the main difference between the two models is that CSDI used Tranformers as diffusion layers within each of its residual blocks while SSSD used the S4 layers as the diffusion layer to capture long-range dependencies. The whole architecture of CSDI and SSSD are shown in Figure 4 and 5 respectively.

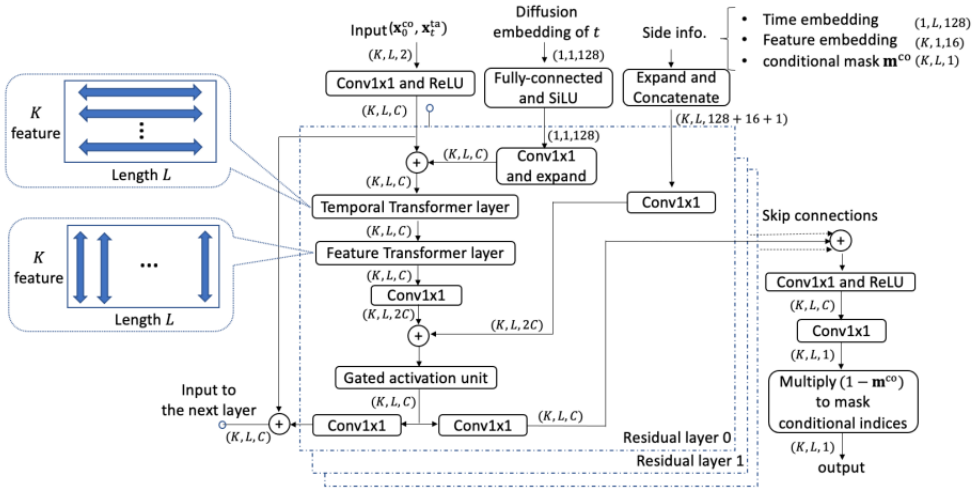

Figure 4: Architecture of ϵ_θ in CSDI [13].

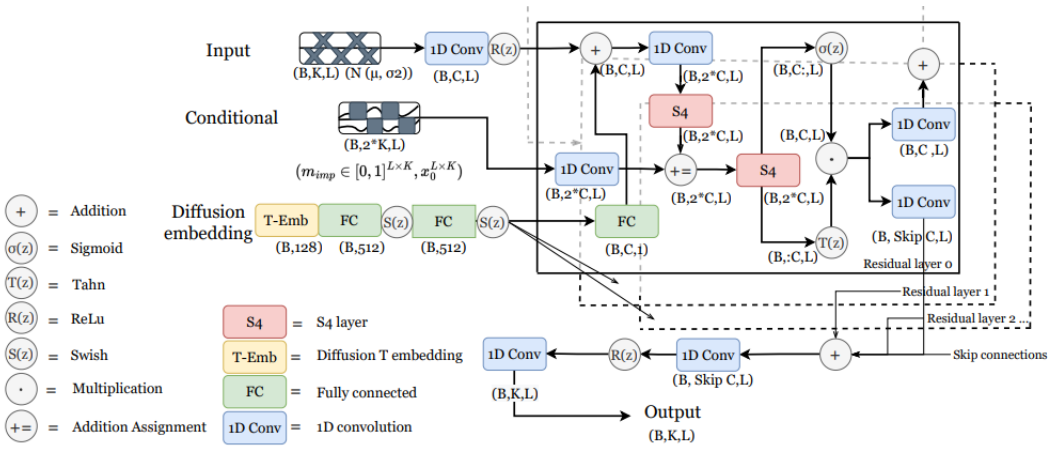

Figure 5: Architecture of ϵ_θ in SSSD [1].

5 Questions: Part 1

5.1 Code Implement and Results Replication

The Tensorflow code for both SSSD and CSDI model is available on github. In the paper [1], the author trained the model for 150000 iterations. However, due to the limitation of the computation power, the SSSD model was only trained for 12000 iterations and the CDSI model was trained for 60000 iterations in this report. Both of the models were trained on the mujoco dataset with 5% missing data, and both models were trained for a same duration. The MSE testing loss for SSSD is 0.7956 and for CSDI is 1.1449. It appears that SSSD outperforms CSDI under a same training duration.

Since the models were not trained for enough iterations, the imputed results do not look as well as it appears in the paper[1,13]. However, to show that the model is actually learning, we plotted the imputed result with the SSSD model at iteration 1200 and iteration 12000 for comparison in Figure 6 and 7. The blue line is the ground truth value and the red points are the imputed values.

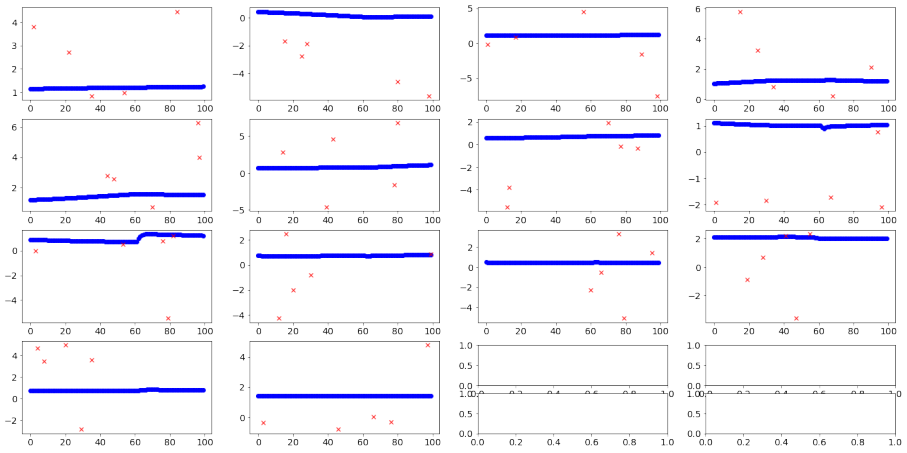

Figure 6: Imputed Results at Iteration 1200

From the figures we can see that the range of the imputed data on y-axis is much smaller and closer to the ground-truth given more iterations, which implies that the model is learning. Therefore, it is reasonable to believe that the model would work better provided that more training time is given.

5.2 Imputation of Stock Price during Holidays

5.2.1 Dataset

All the component stocks of the current Dow Jones 30, EuroStoxx 50 and Hang Seng Index with at least ten years history were downloaded. The dataset contains 2583 days in total, covering all the trading days from 2012-12-01 to 2022-12-01 where there was at least one market open on that day. The data includes 133 stocks, with 29 from Dow Jones 30, 47 from EuroStoxx 50 and 57 from

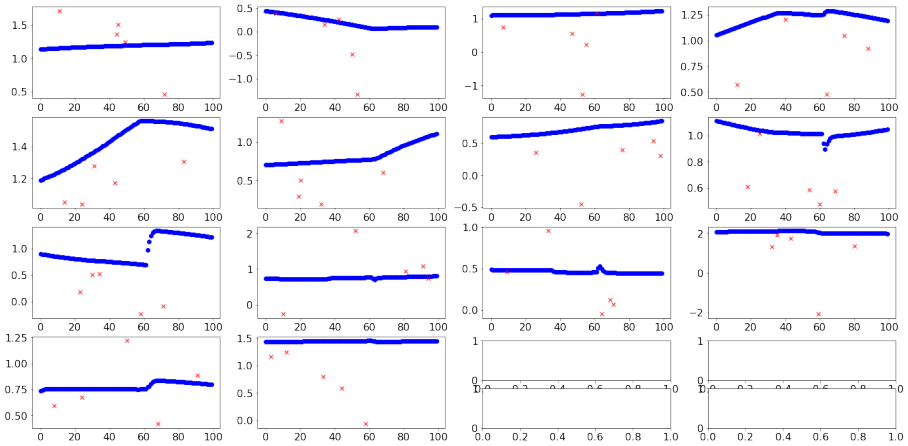

Figure 7: Imputed Results at Iteration 12000

Hang Seng Index. For each stock, the data contains its open price, high price, low price, close price, adjusted close price and volume. Therefore, the shape of the whole dataset is 2583*798.

Among all the 2583 trading days, there are 66 missing days in Dow Jones, 100 missing days in EuroStoxx and 118 missing days in Hang Seng, taking up 2.6%,3.9% and 4.6% respectively.

5.2.2 Assumptions

Before pre-processing the data, we first make some assumptions for the model to apply.

Assumption 1. The return of the stock market follows a normal distribution:

$$\frac{\Delta S(t)}{S(t)} \sim N(\mu \Delta t, \sigma^2 \Delta t)$$

This assumption is quite common in the financial market. Based on this assumption, the stock price change between two different timestamps is related to their time interval. However, the question here is whether the time difference should contain the weekends or holidays, which means whether there is a significant difference between the return during two consecutive trading days during which there is no market open and the return after a holiday during which there is at least one market open.

Assumption 2. The market behaves the same during normal days and the day after weekends, but behaves differently during the days after holidays.

Here, weekends are defined as those when there is no market open, and holidays as those when there is at least one market open. To test this assumption, we did two KS tests. The first one is on the return of the stocks on normal days and the return of the stocks after weekends. The result shows that there is no significant difference between these two samples. (statistic=0.009868, pvalue=0.177520).

The second test is on the return of the stocks on normal days and the return of the stocks after holidays. Under this test, it shows that there is significant difference between these two samples. (statistic=0.065815, pvalue=1.368696e-100)

Based on these two tests, Δt in assumption 1 should contain only holidays where there is at least one market open and ignore the weekends. And this can be simply expressed by adding a time embedding of $s = \{s_{1:L}\}$ to learn the temporal dependency as side information.

Assumption 3. If the holidays were suddenly declared by law not holidays anymore, the behavior of the investors would remain the same.

With this assumption, while imputing the real missing data, we can simply set all the value in the target mask as 0.

Assumption 4. All the missing data are blackout missing, and the missing ratio is 5%.

The missing data in both the DJ and HS market are all blackout missing. However, for EU, it contains both blackout and random missing values.To simplify the problem, we assume that all the missing data are blackout. Since blackout missing data misses more information, if the model can perform well on blackout missing data, it should be able to perform well on random missing data. Furthermore, since the highest missing ratio among the markets is 4.6%, we simply set the missing ratio as 5% for all markets while training the model.

Finally, in order for the SSSD model to apply, we also assume that:

Assumption 5. The stock price function $f_{\leq t} := f(x)|_{x \leq t}$ can be mapped to some polynomial function $g^{(t)}$ where:

(1) the approximation minimizes the error:$||f_{\leq t} - g^{(t)}||_{L2(\mu^{(t)})}$, and

(2) The polynomial $g^{(t)}$ can be mapped to the coefficients c(t) of the basis of orthogonal polynomials defined with respect to the measure $\mu^{(t)}$.

Here $\mu^{(t)}$ can be translated Legendre (LegT), translated Laguerre (LagT) or scaled Legendre measure (LegS), Which are different types of orthogonal polynomials with different recurrent properties and drives to different HiPPO matrix. However, we will later find that this assumption may not hold.

5.2.3 Data Pre-processing

The original data size is 2583*798 as mentioned above. However, due to the computation limitation and convergence problem in high dimension data, we only use the Close Price of all stocks as the feature (K=133) in the later model training process. The data size is thereby 2583*133.

Sliding Window. As a time series dataset, in order to fit it into the diffusion model, we need to first implement a sliding window on the dataset and split the whole dataset into multiple samples. To capture long range dependency, the length of the window L is set to be 100.

Side information. As mentioned in Assumption 1 and 2, we use time embedding of $S = \{s_{1:L}\}$ to learn the temporal dependency.

Observation Mask. Since there are missing values in the data, here, we refer to the method used in [13] and denote an observation mask as $M = m_{1:K,1:L} \in \{0,1\}^{K*L}$ where $m_{k,l} = 0$ if $x_{k,l}$ is missing, and $m_{k,l} = 1$ if $x_{k,l}$ is observed.

Target Mask. To train the diffusion model, the ground-truth missing values is needed. However, in practice, the ground-truth missing values on the holidays are unknown, so we can only make use of the non-missing data and use a binary mask T to distinguish the conditional values (1) from the target values (0). Based on Assumption 4, we randomly set 5% blackout missing values in our target mask T. While calculating the loss, only the value in the target mask are considered.

A sample of the final data is illustrated in Figure 8.

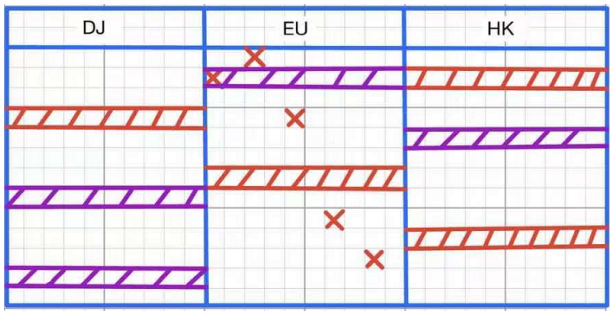

Figure 8: Observation Mask and Target Mask.The red line represents the observation mask and the purple line represents the target mask.

Furthermore, to forecast the price change of HK of the day, the data pre-processing prodcedure is the same as above, except that the data here is the difference between High and Low price, and the corresponding target mask is shown in Figure 9.

In summary, the values of each time series are denoted as $X = \{x_{1:K,1:L}\} \in R^{K*L}$ where $K = 133$ is the number of features and $L = 100$ is the length of time series. We also denote an observation mask as $M = m_{1:K,1:L} \in \{0,1\}^{K*L}$ and a target mask as $T = t_{1:K,1:L} \in \{0,1\}^{K*L}$ to indicate the missing data. We assume time intervals between two consecutive data entries equals to the days of holidays between them where there is at least one market open, and define the timestamps of the time series as $S = \{s_{1:L}\} \in R^L$. Each time series is expressed as {X,M,T,S}. After completing the sliding window, there are 2483 samples in total. We take 1600 as training and the rest as testing data.

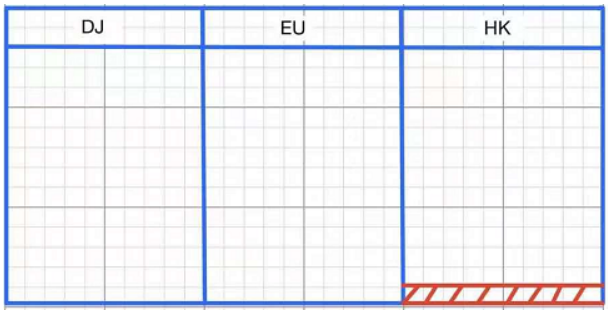

Figure 9: Target Mask for Predicting the Price Change of HK

5.2.4 Model Modification

To fit the SSSD model to our specific data, we modified the structure of the SSSD model by setting the imputation mask as $M * T$, and also concated the side information with the conditional input. For the loss function, we consider only the target mask T here.

The structure of the CSDI model remains the same.

5.2.5 Results

SSSD Model. According to our experiments, the SSSD Model fails to learn on the stock data. The loss fluctuates around 1.0 despite of tuning the learning rate. As described in [1], the model always fails to converge with input channels over 100. We then trained the model with a small subset of the data containing only 9 stocks (3 from each market) to assess the model performance in this case. However, it seems that the model still fails to learn even when there is few channels (see Figure 10).
At the meanwhile, we also tested the data on the Pytorch (the original version) to ensure that the problem does not lie in our code.

CSDI Model. From Figure 11, we can see that the CSDI model is trying to learn on the stock data, which is better than the SSSD model. However, the model training is extremely slow in the situation of 133 stocks, and its performance is also not as well as on the audio datasets.

Regarding the HK market prediction, the result is quite similar, please see Figure 12 below.

6 Discussion

From the result, we can see that the CSDI model outperforms SSSD on the stock market data. The SSSD model fails to learn on the financial data and it looks like it is losing information while training.

There are two possible reasons why the SSSD could not perform well on the stock mark data. The first reason is probably that Assumption 5 is not satisfied. The HiPPO matrix is the funda-

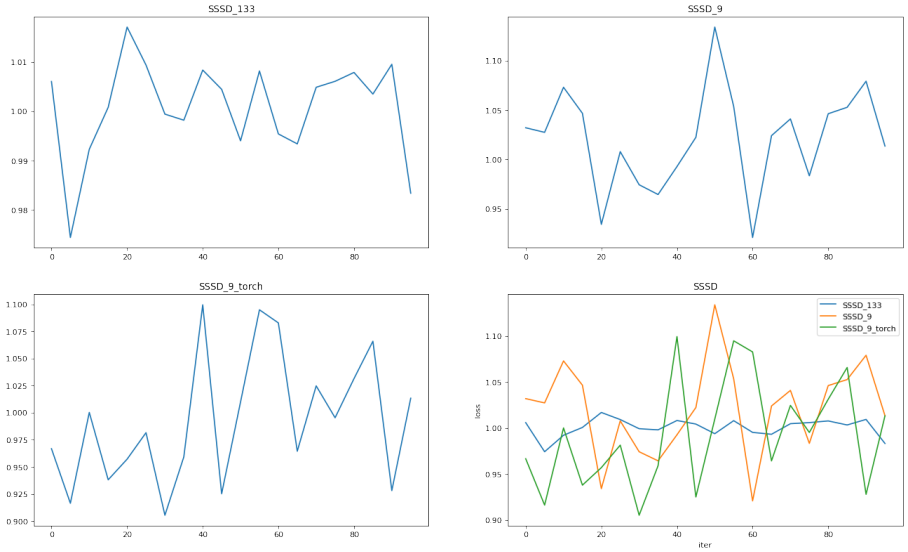

Figure 10: Training Loss on Different Datasets and Model Versions for SSSD

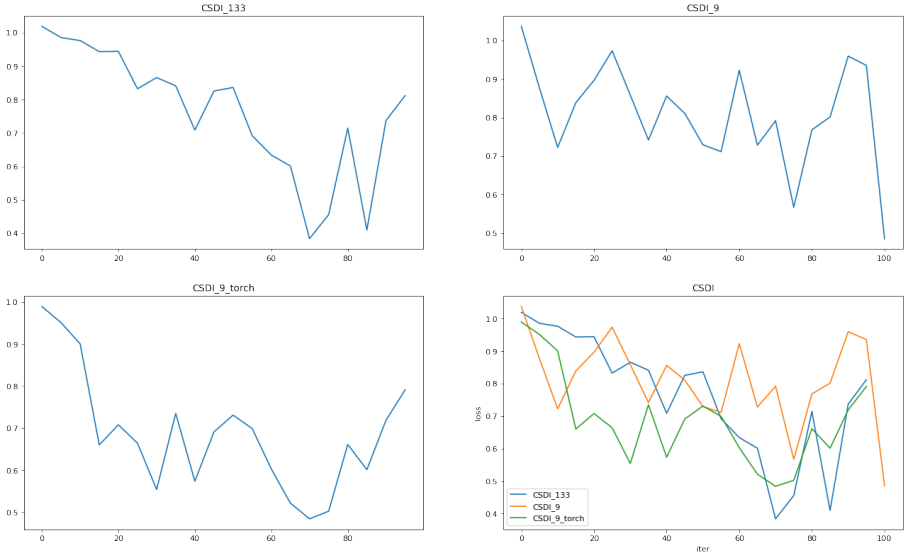

Figure 11: Training Loss on Different Datasets and Model Versions for CSDI

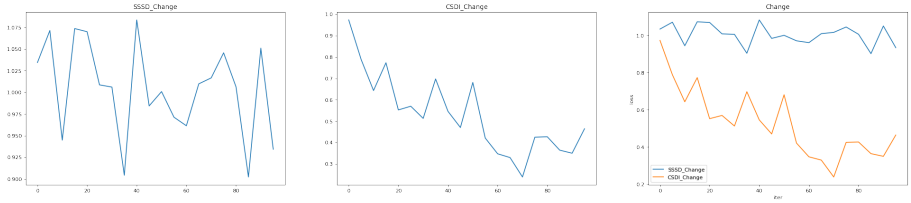

Figure 12: Training Loss of HK Market Prediction for Different Models

mental of the S4 layer. In [4], Gu first approximated the time series f with some polynomial g and then derived the HiPPO matrix based on the properties of g. Therefore, if there's no appropriate g that can simulate f, the S4 layer can not work well.

In the stock market, the stock price is usually assumed to be a Brownian motion where:

$$\frac{dS(t)}{S(t)} = \mu dt + \sigma dB(t)$$

$$(dB)^2 = dt$$

It behaves like a random walk and it is so fluctuated that its quadratic variation equals to dt. Therefore, it's hard to find some proper orthogonal polynomial bases to simulate it, and force to do so may distort the information.

The second possible reason is that while discreting the continuous ODE derived in HiPPO[4] to make it align with the real world computer data, the discretization methods (e.g. forward Euler, backward Euler) are actually approximating the equation based on classic calculus or the first order of the Taylor series. However, these methods are not appropriate to deal with Brownian motion. Instead, it should use the Ito's calculus. And the inappropriate discretization methods may further distort the data.

7 Conclusion

Time series missing data imputing has long been a rich topic in machine learning. In this report, we applied the two most state of the art model, CSDI and SSSD, to the financial stock market and see how it performs. As a result, we found that the CSDI outperforms SSSD in this case.

8 Reference

[1] Alcaraz, J. M. L., & Strodthoff, N. (2022). Diffusion-based time series imputation and forecasting with structured state space models. arXiv preprint arXiv:2208.09399.

[2] Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B., & Bharath, A. A. (2018). Generative adversarial networks: An overview. IEEE signal processing magazine, 35(1), 53-65.

[3] Fang, C., & Wang, C. (2020). Time series data imputation: A survey on deep learning approaches. arXiv preprint arXiv:2011.11347.

[4] Gu, A., Dao, T., Ermon, S., Rudra, A., & Ré, C. (2020). Hippo: Recurrent memory with optimal polynomial projections. Advances in Neural Information Processing Systems, 33, 1474-1487.

[5] Gu, A., Goel, K., & Ré, C. (2021). Efficiently modeling long sequences with structured state spaces. arXiv preprint arXiv:2111.00396.

[6] Gu, A., Johnson, I., Goel, K., Saab, K., Dao, T., Rudra, A., & Ré, C. (2021). Combining recurrent, convolutional, and continuous-time models with linear state space layers. Advances in neural information processing systems, 34, 572-585.

[7] Ho, J., Jain, A., & Abbeel, P. (2020). Denoising diffusion probabilistic models. Advances in Neural Information Processing Systems, 33, 6840-6851.

[8] Kingma, D. P., & Welling, M. (2019). An introduction to variational autoencoders. Foundations and Trends® in Machine Learning, 12(4), 307-392.

[9] Kong, Z., Ping, W., Huang, J., Zhao, K., & Catanzaro, B. (2020). Diffwave: A versatile diffusion model for audio synthesis. arXiv preprint arXiv:2009.09761.

[10] Morrill, J., Salvi, C., Kidger, P., & Foster, J. (2021, July). Neural rough differential equations for long time series. In International Conference on Machine Learning (pp. 7829-7838). PMLR.

[11] Pascanu, R., Mikolov, T., & Bengio, Y. (2013, May). On the difficulty of training recurrent neural networks. In International conference on machine learning (pp. 1310-1318). PMLR.

[12] Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., & Ganguli, S. (2015, June). Deep unsupervised learning using nonequilibrium thermodynamics. In International Conference on Machine Learning (pp. 2256-2265). PMLR.

[13] Tashiro, Y., Song, J., Song, Y., & Ermon, S. (2021). CSDI: Conditional score-based diffusion models for probabilistic time series imputation. Advances in Neural Information Processing Systems, 34, 24804-24816.

[14] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.