TIME SERIES FORECASTING USING GENERATIVE ADVERSARIAL NETWORK

by

Sharon Sone Mamua

May, 2023

Director of Thesis: Rui Wu, PhD

Major Department: Computer Science

Abstract

Time series data is prevalent in many fields, such as finance, weather forecasting, and economics. Predicting future values of a time series can provide valuable insights for decision-making, such as identifying trends, detecting anomalies, and improving resource allocation. Recently, Generative Adversarial Networks (GANs) have been used to learn from these features to aid in time-series forecasting. We propose a novel framework that utilizes the unsupervised paradigm of a GAN based on related research called TimeGAN. Instead of using the discriminator as a classification model, we employ it as a regressive model to learn both temporal and static features. This framework can help generate synthetic data and facilitate forecasting. Our model outperforms TimeGAN, which only preserves temporal dynamics and uses the discriminator as a classifier to distinguish between synthetic and real datasets

TIME SERIES FORECASTING USING GENERATIVE ADVERSARIAL NETWORK

A Thesis

Presented to The Faculty of the Department of Computer Science

East Carolina University

In Partial Fulfillment of the Requirements for the Degree

Master of Science in Data Science

by

Sharon Sone Mamua

May, 2023

Copyright Sharon Sone Mamua, 2023

TIME SERIES FORECASTING USING GENERATIVE ADVERSARIAL NETWORK

by

Sharon Sone Mamua

APPROVED BY:

DIRECTOR OF THESIS:	Rui Wu, PhD
COMMITTEE MEMBER:	Qin Ding, Ph.D.
COMMITTEE MEMBER:	Nic Herndon, Ph.D.
CHAIR OF THE DEPARTMENT OF COMPUTER SCIENCE:	Venkat Gudivada, PhD
DEAN OF THE GRADUATE SCHOOL:	Kathleen Cox, Ph.D.

Table of Contents

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Time-Series data is data collected and/or observed at equivalent time frames from each other. Such data is increasingly used in different sectors such as; finance, geography, weather, and engineering. With such a rise, there is also a need to create models that can predict the outcome for future time steps. There is a lot of historic data that can be used to make informed future decisions using forecasting models. In addition, we want to be able to produce synthetic data that looks very similar to real data for privacy. Analysis can also be conducted on such data to perform informed decisions because real data can be costly and sensitive. Using synthetic data with the same properties and probability distribution as real data is very helpful in forecasting and predictions.

Recurrent Neural Networks(RNNs) have been very prominently used for forecasting. RNNs suffer from the problem of short-term memory. For example, the translation of a long text can have earlier words in a sentence or text overlooked; this can be problematic, as these overlooked words may have great significance in the meaning of the text in future time steps. Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) have been previously proposed as solutions to this problem. Because of their use of hidden states, LSTM and GRU provide a solution for the vanishing of vital features as the neural network goes from one-time step to the next. These will be used as the basic cell units for our discriminator and generator in our proposed model.

Machine learning models are commonly implemented in two forms. First, Supervised Learning is a form of machine learning where that uses labeled data sets and the data sets are trained to accurately predict or classify data sets over time. While unsupervised learning is used to cluster and analyze data sets [1]. In our approach to forecasting time-series data, we will be using both. A Generative Adversarial Network (GAN) is a form of unsupervised learning via generative modeling. In generative modeling, the training data set is obtained from an unknown probability distribution. Then, the model used to generate that data set is learned, then generates a distribution that approximates the training data set as closely as possible [2]. Then, we will predict a one-time step ahead in our data using both synthetic and real data to evaluate the accuracy of our proposed model.

Our Approach is based on the paper Time-series Generative Adversarial Networks [3] where they used a GAN framework to generate synthetic data. They used the generator and discriminator simultaneously, where the discriminator is a classifier that tries to correctly label the data as synthetic or real, while the generator tries to deceive the discriminator. The discriminator is introduced to help improve the quality and integrity of the results produced by the generator. However, in our approach, the discriminator predicts the loss between the synthetic data from the generator and the real data. They both work together simultaneously to minimize the error in the discriminator and generator. Through the combination of unsupervised learning GAN and supervised training in autoregressive models, we are presenting a model for Time-Series forecasting.

The main contributions of this study are : A generative model to produce synthetic data that is very similar to the real data, and in contrast, to the original discriminator using a classifier to distinguish between the real and synthetic data, our discriminator will be a regression model used to predict the loss between the generator and the real data, which will result to the final prediction (one time step ahead) being an addition of the synthetic data(from the generator) to the loss(from the discriminator).

Chapter 2

Related Work

There are numerous studies in the field of time-series forecasting. However, few of these studies utilize GAN with the discriminator as a regressive model. Some of the works detailed in this section will include different research that uses GAN and time-series forecasting in order to give us a look at some previous research that will be useful for the understanding of time-series research.

2.1 Time-series Forecasting Using Recurrent Neural Networks and its variants: Long Short-Term Memory(LSTM) and Gated Recurrent Units(GRU)

Recurrent Neural Networks are types of artificial neural networks, and one of their many advantages is to help with the forecasting of complex data. One of RNNs important features is its ability to recognize patterns in time-series data and make more accurate predictions. Neural networks are based on the biological functionality of neurons. Recurrent Neural Networks consist of hidden neural network units which are connected recurrently. RNNs constitute three components: Input, activation function, and Output [4]. The four common types of activation functions are the Rectified Linear Unit (ReLU), Sigmoid, Tangent hyperbolic (tanh), and Softmax.

One main feature of RNNs is that the same network is used for several time steps; which can tend to serve as memory[5]. However, there comes a problem when faced with a long sequence of data. This is known as the vanishing gradient problem where crucial events which happened several time steps ago may be overlooked. This happens due to the decreasing weights given to data as the time steps go along. This problem has been solved using Long Short Term Memory(LSTM)[6].

LSTMs have an internal state and a number of multiplicative gates: an input gate, a forget gate, and an output gate [7]. LSTMs have dedicated mechanisms for when a hidden state should be updated and also when it should be reset. This helps with the vanishing gradient problem. However, LSTMs have many parameters to be set during the training phase, which requires a large computational power[4]. To help with the computational challenges with large/complex data sets, Gated Recurrent Units(GRU) have been implemented. With GRUs unlike LSTMs, the three gates are replaced by two: the reset gate and the update gate.

2.2 Introduction to Generative Adversarial Networks(GAN)

As previously mentioned, our model is based on Generative Adversarial Networks (GANs). Generative Adversarial Networks (GANs) are a type of deep learning model introduced in 2014 by Ian Goodfellow and his team [8]. GANs are a form of unsupervised learning, where the model learns to generate new data without any pre-existing labels or categories. The GAN architecture consists of two components called Generator and Discriminator. The role of the generator is to generate new data which is similar to the data that is provided, and the role of the discriminator is to differentiate between generated data and real input data. Both synthetic data and real data are used as input to the discriminator. The discriminator needs to correctly classify the input data as real or fake.

Over time, the generator learns to produce increasingly realistic data samples that can fool the discriminator. The end goal is to have a generator that produces synthetic data samples that are indistinguishable from real data samples, while the discriminator becomes increasingly confused and uncertain about which samples are real and which are fake.

GANs have shown promising results in a variety of applications, including image and video generation, music synthesis, text generation, and even drug discovery. GANs have the potential to revolutionize the way we generate and analyze data, and their development continues to be an active area of research in the field of machine learning.

2.3 Time-series Generative Adversarial Networks(TimeGAN)

TimeGAN is a model proposed by Jinsung Yoon et al [3] which is a generative model, trained adversarially and jointly via learned embedding space with both supervised and unsupervised loss. The paper ”Time-series Generative Adversarial Networks” introduces a new type of neural network architecture called Time-series Generative Adversarial Networks (TGANs) for generating synthetic time-series data that is similar to real-world time-series data. TGANs consist of two deep neural networks, a generator and a discriminator, which are trained in an adversarial manner to produce realistic synthetic time-series data.

The generator takes random noise as input and generates synthetic time-series data that is similar to the real data, while the discriminator distinguishes between real and synthetic time-series data. Through iterative training, the generator learns to generate time-series data that can fool the discriminator, while the discriminator learns to distinguish between real and synthetic data.

The paper demonstrates the effectiveness of TGANs on several real-world time-series data and shows that TGANs can generate synthetic time-series data that is comparable to real-world data in terms of statistical properties and predictive accuracy. TGANs have potential applications in areas such as finance, healthcare, and manufacturing, where generating synthetic time-series data can be useful for data augmentation, privacy-preserving data sharing, and predictive modeling.

TimeGAN consists of four network components: embedding function, recovery function, sequence generator, and sequence discriminator. The autoencoding components(embedding and recovery functions) are trained jointly with the adversarial components (sequence generator and discriminator).

The embedding function, e is implemented via a recurrent neural network. The recovery function, r is implemented via a feedforward network at each step. The generator, g takes a random vector, Z from a known distribution. Then generating a tuple of static and temporal features. The generator, g is implemented via a recurrent neural network. The discriminator, d operates in the embedding space where it receives static and temporal features and classifies them as real or fake. The discriminator, d is implemented via a recurrent bidirectional network with a feedforward output layer.

The first objective is for the embedding and recovery functions to enable the reconstruction of the original data. They provide a function called reconstruction loss. During training, the objective is to minimize this loss. In TimeGAN proposed method, the generator is exposed to two types of inputs during training. The generator receives synthetic embeddings in order to generate the next synthetic vector. Gradients are then computed to calculate the unsupervised loss. The main objective during this step is to allow maximizing the efficiency of the discriminator and minimize the efficiency of the generator. The generator does not rely mainly on the discriminator's binary adversarial feedback but also receives sequence embeddings of the actual data. This helps generate the next latent vector. This model yields the supervised loss.

Traditional time-series forecasting techniques, such as ARIMA and exponential smoothing, rely on statistical methods and historical data to forecast future values. These techniques often assume that the underlying data is stationary and linear, which may not be true in many real-world scenarios. GANs offer an alternative approach to time-series forecasting by generating synthetic data that captures the underlying distribution of the data.

The authors evaluate their proposed method, TimeGAN with RCGAN[9] and C-RNN-GAN[10], the two most closely related methods. They calculated a discriminative and predictive score. For the discriminative score, they trained a classification model to distinguish between the real and synthetic data. Then, reported the classification error. However, since our proposed model doesn't utilize the discriminator as a classification model we are not going to utilize those scores. On the other hand, the predictive score was obtained by training a post-hoc sequence prediction model to predict the temporal vectors for the next time step. The model was trained on the synthetic data, then evaluated using the original data. Performance was evaluated using the mean absolute error (MAE), which we are going to use as well to evaluate our proposed model.

Chapter 3

Proposed Method

In this thesis, we have created a framework for time-series forecasting using a generative adversarial network. Within this framework, there is a generator and discriminator as the two main components for predictions. Where the generator is the model used to generate synthetic data, and the discriminator is a model used to predict the loss between the synthetic data and real data at every time step. These two models are used simultaneously to produce synthetic data. They are later trained in post-hoc models to predict the next time step in the original data.

Further, we utilize recovery and embedding functions. Our proposed model for generating synthetic data comprises four components: recovery and embedding functions, generator, and discriminator, a model previously proposed by Yoon [3]. The recovery and embedding functions are autoencoding components used to accurately take the data between the feature and latent spaces which enables the learning of the dynamic temporal features of the data. As previously mentioned, our main contribution which differs from the TimeGAN paper[3] is that instead of using our discriminator as a classifier to differentiate between the synthetic and real data, we use the discriminator to predict the difference between the values in the real and synthetic data. Then, we add this predicted loss to the generated data to predict the record for the next time step.

First, we develop synthetic data by jointly training the generator and discriminator. As previously mentioned the embedding and recovery functions are also used to optimize the quality of the synthetic data. Then, to test the generator's and discriminator's efficiency, we use a post-hoc sequence-prediction model (by optimizing a 2-layer LSTM) to predict next-step temporal vectors over each input sequence.

The following sections will discuss the architecture of the proposed framework. We will detail the generation of the **Synthetic Data Generation** methodology and **Post-hoc Prediction Model** used in our proposed method.

3.1 Architecture Diagram

The overall setup of this time-series forecasting model relies on the design of a generative adversarial network(GAN). The architecture diagram given below illustrates our proposed framework.

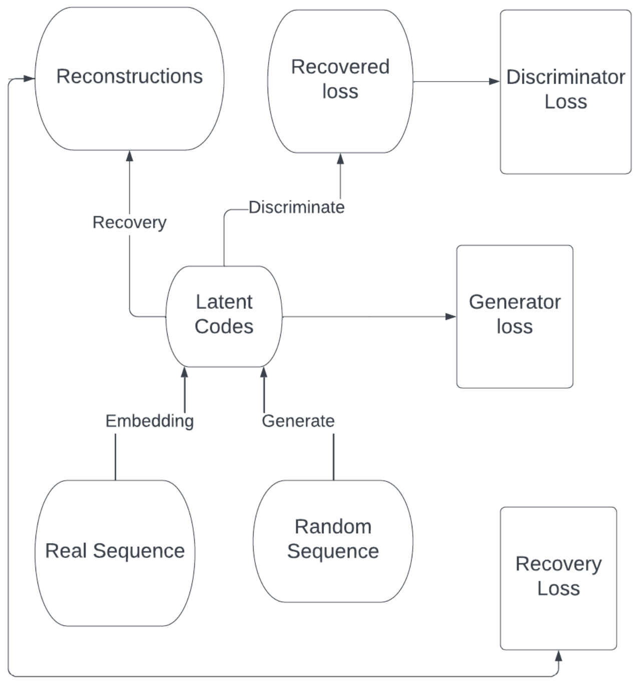

Figure 3.1: Architecture diagram of component functions and objectives. Latent Codes are generated by using an autoencoding network. Codes are reconstructed, and the loss between both (real and reconstructed data) is minimized in training.

Figure 3.1 shows how the data is transformed into latent codes allowing the adversarial network to learn the underlying temporal dynamics of the data via lower-dimensional representations. Time-series data usually contains temporal correlations, which cannot be directly captured and interpreted by traditional image-focused disentangle methods like LSTM[11], Auto-Regressive Integrated Moving Average (ARIMA) [12]. Hence, representation learning in the time-series setting primarily deals with the benefits of learning compact encodings for the use of downstream tasks such as forecasting. Hence in our method, latent space helps fix this problem.

3.2 Synthetic Data Generation

One of the two main steps is generating synthetic data that is very similar to the original data. To achieve this, we use the GAN framework. This is made up of four components: embedding and recovery functions, generator, and discriminator.

3.2.1 Embedding and Recovery Functions

The embedding function takes the original features of the data into latent spaces. This function is implemented with the help of a feedforward neural network at each step. Which returns the static and temporal features of the real data.

The recovery function takes the data from latent spaces to original spaces. This function is used to train the recovery of data from both its static and temporal features.

3.2.2 Reconstruction Loss and Embedding loss

A reversible mapping between original and latent spaces is required. Hence, the embedding and recovery functions should enable accurate reconstructions.

To achieve this, we put parameters in our model to minimize the loss between recovered data and real data using the Mean Squared Error.

$$E\_loss = tf.losses.mean\_squared\_error(X, Xtilde) \tag{3.1}$$

3.2.3 Generator and Discriminator

The generator is the main model responsible for producing the synthetic data. This model takes a random vector sequence with a similar probability distribution to the original data. First, the generator takes the data to latent space, then calculates the unsupervised loss and tries to minimize this loss in every iteration. This is done with the help of two models: a *generator*, a generator network for the temporal features, and $supervisor1$, for static features.

The discriminator operates in the latent spaces where it tries to predict the loss between the generator output and the original data. The discriminator has a supervisor as well used to get the static features. The discriminator consists of *discriminator* and $supervisor2$.

3.2.4 Generator and Discriminator Losses

First, for the generator loss, we access the difference between X, real data, and X_hat, the generator data at that time step.

$$G\_loss\_U = tf.losses.absolute\_difference(X, X\_tilde)$$

G_loss_U is the loss between $X$and X_tilde, recovered real data. Then, we get a supervised loss, G_loss_S, which calculates the mean squared error of the data in latent space. H, which is the result of embedding the real data into latent space, and $H\_hat\_supervise$, which results from H being run on the generator's supervisor.

$G\_loss\_S = tf.losses.mean\_squared\_error(H[:, 1:, :], Hhatsupervise[:, :-1, :])$

Then calculate two moments:

$$G\_loss\_V = G\_loss\_V1 + G\_loss\_V2$$

Final generator loss equation:

$$G\_loss = G\_loss\_U + 100 * tf.sqrt(G\_loss\_S) + 100 * G\_loss\_V \tag{3.2}$$

Next, we have the discriminator loss which evaluates the loss between the synthetic data plus discriminator loss and real data

$$D\_loss = tf.losses.mean\_squared\_error(X, Y\_hat + X\_hat) \tag{3.3}$$

3.2.5 Jointly Training

This proposed model is heavily reliant on the joint training of both the generator and discriminator. Where the generator is able to predict data as close to the real data, and the discriminator is able to predict the loss between the real data and generated data at every iteration. The objective is to minimize the discriminator loss(the difference between the generated output and real data), and generator loss to get the generated data to be as close to the real data as possible.

The model is run for 10000 iterations. The resulting generated data is X_hat, which is the output from the discriminator.

3.3 Post-hoc Prediction Model

At this stage, we have generated the synthetic data, now is the time for the prediction of time steps ahead.

First, we get a predictive score that uses a sequence-prediction model to predict next-step temporal vectors over each input sequence. This model consists of a Multi-Layer recurrent neural network made up of gated recurrent units(GRU) as neurons. The model predicts the next time step in the input data. The model is trained on the synthetic data, and tested on the real data.

Then, we get our novel prediction model which predicts the loss between the synthetic data and the real data. This model also consists of a Multi-Layer recurrent neural network made up of gated recurrent units(GRU) as neurons. Then, evaluates the final prediction as the sum of generated data and the predicted loss from the discriminator.

Chapter 4

Implementation

4.1 Application Work Flow

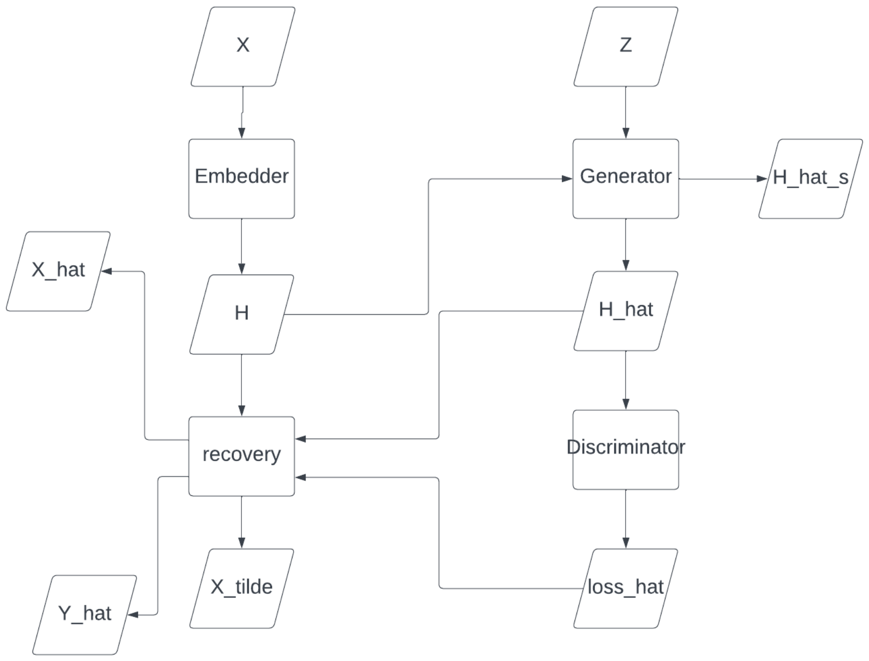

Figure 4.1: Training scheme for generating synthetic data in the proposed model. Embedder gets real data and transforms it into latent codes. Recovery takes the latent codes to the original data. The generator learns from latent codes of random sequence to predict data. Discriminator learns from real and synthetic data to predict the loss(real data - prediction from generator)

Figure 4.1 shows the training scheme for generating synthetic data in the proposed model. This figure is used to visually represent the roles of the different components which will be described in-depth below.

4.2 Generating Synthetic Data

For the first part of our model which is generating the synthetic data, we start by constructing placeholders for our data; X, Z, T

X = tf.placeholder(tf.float32, [None, max_seq_len, dim], name = ”myinput_x”)

Z = tf.placeholder(tf.float32, [None, max_seq_len, z_dim], name = ”myinput_z”)

T = tf.placeholder(tf.int32, [None], name = ”myinput_t”)

X represents the input data, which has dimensions similar to original data grouped by sequence. T represents the time data. Z represents the random sequence data.

We use Python code defining a function named TimeGAN, which generates synthetic time-series data using the TimeGAN algorithm. The TimeGAN algorithm consists of four networks: an embedding network, a recovery network, a generator network, and a discriminator network.

The function takes two arguments: oridata, the original time-series data used as a training set for generating synthetic data, and parameters, a dictionary containing TimeGAN network parameters, including the number of hidden layers, number of iterations, batch size, etc. These hyperparameters will be discussed in the next section.

The function begins by resetting the default TensorFlow graph. Next, the original data is normalized using the MinMaxScaler function. Then, the TimeGAN networks are defined using helper functions: embedded, recovery, supervisor1(for generator), and supervisor2(for discriminator).

The embedder function takes the original time-series data and returns the latent space embeddings of the data. The recovery1 and recovery2 functions reconstruct the original time-series data from the embeddings. The supervisor1 and supervisor2 functions generate a new sequence based on the previous sequence using the latent representations generated by the generator and discriminator simultaneously.

e_vars= $[v for v in tf.trainable\_variables() if v.name.startswith('embedder')]$
$r1\_vars$ = [v for v in tf.trainable_variables() if v.name.startswith('recovery1')]
E_solver = tf.train.AdamOptimizer().minimize($E\_loss, var\_list = e\_vars + r1\_vars)$

$sess.run([E0\_solver, E\_loss\_T0], feed\_dict = X : X\_mb, T : T\_mb)$

Finally, the synthetic data is generated using the TimeGAN algorithm, which involves training the generator and discriminator networks simultaneously for a number of iterations until the synthetic data matches the original data. The generated synthetic data, X_hat is returned by the function.

4.3 Post-hoc Training and Testing

4.3.1 Predictive Score

This is implemented using Python code for evaluating the performance of a predictive model using post-hoc RNN time steps ahead prediction. Specifically, it takes as input the original data and the generated synthetic data and outputs the mean absolute error (MAE), RMSE, and R2 scores of the predictions and the original data.

The code builds a post-hoc RNN predictive network using the generated synthetic data and then tests the trained model on the original data. The performance is evaluated in terms of MAE, root mean squared error (RMSE), and R-squared (R2) score.

The code imports necessary packages including TensorFlow, NumPy, matplotlib, and sci-kit-learn. It defines the predictive score metrics function that takes in two arrays of data real data and generated data. It then initializes the

TensorFlow graph and sets basic parameters such as the maximum sequence length, the sequence length of each time series, and the number of dimensions. The function builds a post-hoc RNN predictive network with a simple predictor function that uses a GRU cell and a fully connected layer with a sigmoid activation function. The loss function for the predictor is the absolute difference between the predicted and true values.

The function then trains the predictor using the generated synthetic data and tests the trained model on the original data. It computes the MAE, RMSE, and R2 scores for each time series.

4.3.2 Discriminative Score

The section is where we compute our final prediction using our proposed method that utilizes the prediction of the loss between the synthetic and real data.

This function takes inputs $(ori\_date, generated\_data)$, and we compute a new data set, $new\_prediction = ori\_data - generated\_data$, which is the loss between the synthetic and real data. We split the data into 70% : 30% training/testing split which are chosen at random. We train an RNN predictive network with a simple predictor function that uses a GRU cell and a fully connected layer with a sigmoid activation function to predict one step ahead in $new\_prediction$.

Our final prediction is calculated as the sum of the loss and the generated synthetic data.

This is the input data for the model. This data is for the time step, t.

$$X\_mb = list(ori\_data[i][: -1, : (dim - 1)] for i in test\_idx)$$

$T\_mb = list(ori\_time[i] - 1$ for i in $test\_idx)$

$y\_pred\_curr = sess.run(y\_pred, feed\_dict = X : X\_mb, T : T\_mb)$

This is the generator data for one time step ahead, t+1

$Y\_mb = list(np.reshape(generated\_data[i][1 :, (dim{-}1)], [len(generated\_data[i][1 :, (dim - 1)]), 1])$ for i in $test\_idx)$

Then the generated synthetic data is added to the predicted loss,

$$new\_y\_pred = Y\_mb + y\_pred\_curr \tag{4.1}$$

All predicted datasets are flattened to $[None, max\_seq\_len - 1, 1]$ before training/testing in order to help speed up training by reducing the dimensions.

4.4 Hyperparameters

Our hyperparameters for the execution of our code are as follows:

- –data_name: This refers to the name of the data set, which is one of the data described in 5.1.
- –seq_len: This refers to the number of time-series records grouped as one time step. This varies from data to data depending on the time interval the data was collected. So we can want to group the records per hour, per day, per month, etc.
- –module: This refers to the name of the units that will be used in the Recurrent Neural Network units for our different models: embedding function, recovery function, generator, and discriminator.
- –hidden_dim: This refers to the number of units used in each layer(num_layer) of the recurrent neural network.
- –batch_size: This refers to the number of training examples used in one iteration of the training.
- –iteration: This refers to the number of cycles used in training the model.

Here is the code specifying the hyperparameters on the command line. The sequence length is the only parameter that changes by data because we want to consider one time step by hour or day.

```
python main_TimeGAN.py –data_name energy –seq_len 6 –module gru –hidden_dim 24 –num_layer 3 –iteration 10000 –batch_size 128 –metric_iteration 1

python main_TimeGAN.py –data_name stock –seq_len 5 –module gru –hidden_dim 24 –num_layer 3 –iteration 10000 –batch_size 128 –metric_iteration 1

python main_TimeGAN.py –data_name electricity –seq_len 24 –module gru –hidden_dim 24 –num_layer 3 –iteration 10000 –batch_size 128 –metric_iteration 1

python main_TimeGAN.py –data_name solar –seq_len 24 –module gru –hidden_dim 24 –num_layer 3 –iteration 10000 –batch_size 128 –metric_iteration 1

python main_TimeGAN.py –data_name sine –seq_len 10 –module gru –hidden_dim 24 –num_layer 3 –iteration 10000 –batch_size 128 –metric_iteration 1
```

Chapter 5

Experiment Results and Discussion

This work proposes a framework to predict time-series data for one time step ahead, utilizing a generative adversarial network framework(GAN) for synthetic data generation and using set data for said predictions. Although previous studies utilize GAN, our study utilizes the discriminator as a regressive model to predict the loss between the generated and real data. This work utilized GAN to generate synthetic data, and a post-hoc time-series regression model (by optimizing a 2-layer LSTM) to predict the difference between the sequences from the original and generated data (Discriminative Score). Then, train a post-hoc sequence-prediction model (by optimizing a 2-layer LSTM) to predict next-step temporal vectors over each input sequence. Then, we evaluate the trained model on the original data(Predictive Score). The models will be evaluated using the mae, r^2, and $rmse$ metrics.

r^2 (r-squared) signifies how much of the resulting variable data can be explained using the other variables. In particular, the predictors (variable information used in the prediction of the machine learning algorithm) are compared to the resulting variable. The proportion of the variance between these variables is called r^2. It provides an indication of how well the model fits the data, **with higher values indicating a better fit**. R-squared is also easy to interpret, as it represents the proportion of the variation in the target variable that can be explained by the model.

However, R-squared has some limitations. It can be sensitive to the number of variables in the model, and it doesn't provide information on the accuracy of individual predictions.

Similarly, the *rmse* (root mean squared error) metric illustrates how far the resulting predictions are from the observed result variable. In simpler terms, *rmse* directly portrays how close the model predictions compare to the actual observations from the initial data. Further, this metric can be considered absolute in that it is a direct comparison between the predicted and actual measurements.

Mean Absolute Error (MAE) is a popular metric used to evaluate the performance of regression models in machine learning. MAE measures the average absolute difference between the predicted and actual values of a target variable. **The smaller the MAE value, the better the model's performance**.

Like MAE, RMSE measures the difference between predicted and actual values of a target variable, but it gives more weight to larger errors.

All of these metrics are important for the evaluation of machine learning algorithms. Each one illustrates not only how well the machine learning algorithms compare in terms of model fit, but also in the behavior of the data.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=0}^{n} Y_i^2 - Y_{hat}{}_i^2} \tag{5.1}$$

$$MAE = \frac{1}{n}\sqrt{\sum_{i=0}^{n} |Y_i - Y_{hat\,i}|} \tag{5.2}$$

where Y_i are the observed values and $Y_{hat\,i}$ are the predicted values at i.

$$r2 = 1 - \frac{\sum_{i=0}^{n}(Y_{hat\,i} - Y_i)^2}{\sum_{i=0}^{n}(Y_i - Y_{bar})^2} \tag{5.3}$$

y_{hat} represents the predicted values, y_i represents the observed values and y_{bar} represents the mean of all the values

5.1 Data

We are going to evaluate our proposed method on a variety of characteristic benchmark time-series data. The following data are publicly available on the UCI machine learning repository[13], and have been used in a variety of time-series research we can compare our model with.

- **Sine.** We simulate multivariate sinusoidal sequences of different frequencies, f, and phases θ, providing continuous-valued, periodic, multivariate data [3].
- **Solar.** Solar data has 137 records with 52560 instances.
- **Energy.** This data is collected every 10 minutes for about 4.5 months. The house temperature and humidity conditions were monitored with a ZigBee[14] wireless sensor network. The timer was set at 10 minutes for 4.5 months. They have 28 records and 19735 attributes. This data can be found at this link
- **Electricity.** This is a time-series data collected every 15 minutes for 370 records/clients. There exist 140256 instances of these records. This data can be found at this link
- **Stock** This is stock data from Yahoo Finance. Stock data is recorded daily during weekdays.

5.2 Results

Table 5.1: Mean Absolute Error(MAE) results for comparison of the Original TimeGAN Model[3] against our proposed model using the discriminator as a regressive instead of a classification model. The table is organized by data.

Data	Original GAN	Proposed GAN	
		Prediction (w/o d loss)	Prediction (with d loss)
Energy	0.3017	0.2852	**0.20684**
Stock	**0.0393**	0.08773	0.0532
Electricity	**0.0379**	0.0388	0.09283
Solar	0.0408	**0.0293**	0.145
Sine	0.1241	0.06942	**0.06282**

Table 5.2: Root Mean Squared Error(RMSE) results for comparison of the Original TimeGAN Model[3] against our proposed model using the discriminator as a regressive instead of a classification model. The table is organized by data.

Data	Original GAN	Proposed GAN	
		Prediction (w/o d loss)	Prediction (with d loss)
Energy	0.3964	0.33108	**0.26374**
Stock	0.0590	0.08773	**0.0552**
Electricity	**0.04435**	0.04716	0.11264
Solar	0.0611	**0.04024**	0.15617
Sine	0.13791	0.07943	**0.07135**

Table 5.3: R-Square(R-2) results for comparison of the Original TimeGAN Model[3] against our proposed model using the discriminator as a regressive instead of a classification model. The table is organized by data.

Data	Original GAN	Proposed GAN	
		Prediction (w/o d loss)	Prediction (with d loss)
Energy	-1.1407	-2.49607	**-0.92776**
Stock	**-1.6961**	-478.00693	-1012.46742
Electricity	-0.5563	**-0.16531**	-12.53315
Solar	-12.736	**-11.11944**	-13693.53011
Sine	-26157.72	-25761.89194	**-30033.24543**

Table 5.4: Root Mean Squared Error(RMSE) results for comparison of the proposed discriminator and the actual loss between the synthetic data and real data

Data	Proposed GAN
Energy	0.1085
Stock	0.07676
Electricity	0.04799
Solar	0.0920
Sine	0.10113

5.3 Discussion

First, the *rmse* and *mae* results are very similar. For both the energy and sine data our proposed method worked best. However, with the solar data, just a partial part of our proposed method worked. Even though our predicted score, which is a result of using synthetic data generated with a regressive discriminator, than the original TimeGAN[3], the addition of the predicted loss(from the discriminator) did not improve the overall prediction of times step ahead.

However, for both the stock and electricity data, our proposed method did not help much in data prediction. Also, these were the only data for which we had a worse overall score than the predictive score(prediction without the addition of the loss from the discriminator).

Table 5.4 refers to the *rmse* scores for the loss between the real data and synthetic data, and the output from the discriminator model which predicts the former. The error between the two is small which should help in the final prediction. However, this is not the case for both electricity and solar data.

5.4 Constraints

One requirement for this model to be most efficient is the data collection methods should be standardized to ensure the data is collected at the same intervals.

Along with this proposed framework, there are some constraints regarding the implementation. First, for some of the data, there wasn't information about the time interval. Hence, during the tuning of the hyperparameters for the sequence length we were not able to immediately find the sequence that would be best to get the best results. This is the possible reason for the outcome of the results in the electricity and stock data.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this study, we proposed a model for generating synthetic time-series data and time-series prediction using the TimeGAN algorithm. The generated synthetic data was evaluated using post-hoc RNN one-step ahead prediction, where the performance was evaluated in terms of MAE, RMSE, and R-squared score, which proved to outperform some existing models. The proposed model can be utilized for generating synthetic data in various fields, including finance, energy, and climate modeling, among others. This study contributes to the development of time-series data generation and can help to fill the gaps in existing data, which can be used for various applications, including training deep learning models.

6.2 Future Work

First, we need to test the model with more time series data. This will help with checking if there is an improvement in results with only certain time series data sets, and that will help us better our model for others. We will need to test our model against other state-of-the-art models: Graph Neural Networks, ARIMA, GRU, and LSTM.

Also, we ran into a bottleneck when predicting loss because when getting the difference between two quantities we inevitably ran into negative values which we believe tend to confuse the model. Eventually, we will need to find a way to normalize the loss that works for our model.

BIBLIOGRAPHY

[1] "Supervised vs. unsupervised learning: What's the difference?" [Online]. Available: https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning

[2] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. X. abd David Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," vol. 63, no. 20, Oct 2020.

[3] J. Yoon, D. Jarrett, and M. van der Schaar, "Time-series generative adversarial networks," *NeurIPS Proceedings*, 2019.

[4] K. Sako, B. N. Mpinda, and P. C. Rodrigues, "Neural networks for financial time series forecasting," *Entropy*, vol. 24, no. 5, p. 657, 2022.

[5] K. Benidis, S. S. Rangapuram, V. Flunkert, Y. Wang, D. Maddix, C. Turkmen, J. Gasthaus, M. Bohlke-Schneider, D. Salinas, L. Stella, F.-X. Aubet, L. Callot, and T. Januschowski, "Deep learning for time series forecasting: Tutorial and literature survey," *ACM Computing Surveys*, vol. 55, no. 6, pp. 1–36, dec 2022. [Online]. Available: https://doi.org/10.1145%2F3533382

[6] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 11 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[7] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, "Dive into deep learning," *arXiv preprint arXiv:2106.11342*, 2021.

[8] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[9] C. Esteban, S. L. Hyland, and G. Rätsch, "Real-valued (medical) time series generation with recurrent conditional gans," 2017. [Online]. Available: https://arxiv.org/abs/1706.02633

[10] O. Mogren, "C-rnn-gan: Continuous recurrent neural networks with adversarial training," 2016. [Online]. Available: https://arxiv.org/abs/1611.09904

[11] Y. Li, Z. Chen, D. Zha, M. Du, D. Zhang, H. Chen, and X. Hu, "Learning disentangled representations for time series," 2021. [Online]. Available: https://arxiv.org/abs/2105.08179

[12] S. Masum, Y. Liu, and J. Chiverton, "Multi-step time series forecasting of electric load using machine learning models," *Artificial Intelligence and Soft Computing*, p. 148–159, 2018.

[13] [Online]. Available: https://archive.ics.uci.edu/ml/index.php

[14] "Zigbee," Mar 2023. [Online]. Available: https://en.wikipedia.org/wiki/Zigbee