Generative Adversarial Network for generating synthetic time series data and evaluation

## 3.1. Data Augmentation

Generative adversarial networks (GANs) are a type of AI algorithm created to address the challenge of generative modeling. The primary objective of a generative model is to analyze a set of training data and understand the probability distribution that produced those examples. There are instances in medical and financial industry where full access of data is difficult to achieve due to data privacy and other GDPR issues. These challenges led to the use of GAN models to produce synthetic data which can be reused to replicate the actual data. Supervised learning, by definition, depends on human oversight to supply an output example corresponding to each input example. A significant drawback is that current supervised learning methods frequently need millions of training examples to surpass human performance, while a human might achieve satisfactory results with only a few examples. To minimize the need for extensive human supervision and reduce the number of examples necessary for learning, many researchers are now focusing on unsupervised learning, often employing generative models (I.Goodfellow., 2016).

Before dwelling into the architecture, lets understand more about generation of synthetic data. synthetic data refers to computer-generated graphics used to train computer vision models. Synthetic data is an information that is artificially generated rather than events produced in the real world. It has multiple uses in production and is increasing in popularity (Nikolenko., 2021). In this case, there are a list of stock prices for companies over the last 45 years but a synthetic Using this dataset, a synthetic data will be created and stored in the result path with the name TimeSeriesGAN.h5 format. A .h5 format is typically associated with HDF5 (Hierarchical Data Format version 5), a file format and set of tools designed to store and organize large amounts of data. In machine learning, .h5 files are often used to save trained models. For instance, TensorFlow and Keras, popular machine learning libraries, use the .h5 format to save and load models, as it efficiently stores the model architecture, weights, and optimizer state in a single file. The real dataset used is a structured dataset, hence, the GAN methodology will be focused on achieving a desirable synthetic data suitable for Supervised Learning. The objective of supervised learning is fairly simple to define, and all supervised learning algorithms share a common aim: to learn how to correctly match new input examples with their corresponding outputs. For example, an object recognition algorithm might classify a photo of a dog by associating it with a label such as "DOG." (Goodfellow et al., 2020)

To summarise, synthetic dataset is a computer generated and its derived from the real dataset using models and algorithms to replicate the properties and characteristics of real world data.

## 3.2. Development of GAN Architecture

**General Architecture**

Basic GAN algorithms are grounded in game theory and operate within a framework where two neural networks compete with one another. While this competitive architecture has achieved remarkable results compared to earlier generative models, a significant challenge remains: GANs are notoriously difficult to train. Consequently, researchers are exploring new methods to enhance the original GAN architecture (Ghosh et al., 2020).

A diagram of a data processing process

Description automatically generated

GAN consists of 2 models that compete each other to analyse, capture, and copy the variations within a dataset. Those 2 models are Generator and Discriminator. As illustrated in the above figure, Generator will generate a dataset and discriminator will distinguish if the data is real or fake. This fake dataset is generated by developing a random noise close to the actual environmental factors. Neural networks (CNN, RNN, LSTM) are used to run the random series on Generator model to develop the data. The aim of the generator is to create dataset whereas the discriminator model aims to classify if the synthetic data is real or different from the actual sample data.

The discriminator network penalizes the generator for failing to deceive the discriminator. The Fine Tuning (training) which can be also termed as “Back Propagation” is used to adjust each weight in the right direction by calculating the weights impact on the output. It is used to obtain the gradients that help to change the generator weights. The ultimate goal is to reach a point where the generator produces flawless replicas of the input domain samples, and the discriminator is unable to differentiate between real and synthetic data. Once both models are thoroughly trained and no longer capable of further improvement, it is said that the network has reached a Nash equilibrium. However, achieving this equilibrium is a challenging task, as it is more complicated than optimizing a single objective function.

**Types of GAN Architecture**

A few types of GAN architecture used for referencing for producing time series synthetic data are

1. Vanilla GAN - A Vanilla GAN is a basic form of a generative adversarial network where both the Discriminator and Generator are constructed as simple multilayer perceptron. It adheres to the standard GAN architecture. The algorithm is straightforward, utilizing a simple mathematical model with a gradient descent approach. The core structure of a Vanilla GAN involves feeding noisy data into the Generator, while the Discriminator's role is to classify whether the generated samples are real or fake (Durgadevi et al., 2021).

The expression for Vanilla GAN is as follows

(X-P data(X) [log D(x)] + EZ- p(Z) [log (1-D(G(Z)))]

Where, P data (X) – Distribution of real data

P(Z) – Distribution of Generator

X - Sample from P data (X)

Z – Sample from P(Z)

D(X) – Discriminator Network

G(Z) – Generator Network

During the training of the generator in a GAN, the component of the loss function that involves real data is omitted. The discriminator's parameters are updated based on the loss function that takes into account both real and generated samples. In contrast, the generator's parameters are updated based solely on the loss function related to the generated data. This means the generator focuses on improving the quality of the synthetic data to deceive the discriminator, while the discriminator learns to distinguish between authentic and synthetic samples.

1. Conditional GAN

A Conditional GAN (CGAN) is another supervised learning variant of GANs, where both the Discriminator and Generator are conditioned on additional auxiliary information denoted as 'C'. In this setup, the Generator receives both the input noise 'Z' and the auxiliary data 'C', which are combined and trained together, as illustrated in Fig. This approach allows the model to be flexible in how it uses the hidden representation 'C', which typically includes class labels or data from different modalities. The network structure commonly used for CGANs is a MultiLayer Perceptron (MLP)

A diagram of a fake model

Description automatically generated

1. Wasserstein GAN

Considering that the discriminator was perfectly trained, the Jensen-Shannon Divergence (JSD) comes into picture as distance between Generator’s distribution P(Z) and real data distribution P data(X). The JSD is a balanced and smoothed version of Kullback- Leibler divergence for assessing the similarities between these two probability distributions. JSD can effectively gauge how closely the generated data resembles the actual data. To address the limitations of the Jensen-Shannon Divergence (JSD) when dealing with singular measures, the Wasserstein metric was introduced as an alternative method for quantifying the distance between the Generator's distribution and the real data distribution.

Even though, this approach provides a better measure compared to Vanilla GAN, Wasserstein GAN is more applicable for generating realistic samples image distributions. The Wasserstein metric used in WGANs is based on a notion of distance between individual images, which induces a notion of distance between probability distributions of images. For the current scenario of structured data, Vanilla GAN can be used as a good reference for architecture (Adler et al., 2018).

**Components of GAN**

The architecture of the synthetic GAN as shown in above figure implements the collaborations of a GAN and different components, where G (Z) is the generator and D (X)is the discriminator. But an additional supervised applied to the latent space can lead to a step-by-step supervised loss depending on the distribution of the original data. This helps the model learn from the transition dynamics in authentic sequences. Post training, the generator produces synthetic samples which are then processed by the classification component.

Yoon et. al (2019) introduced a TimeGAN model with 4 network components namely, an embedding function, recovery function, sequence generator and sequence discriminator. The objective of the time GAN framework is to lead a training process that incorporates an additional supervised loss to guide the adversarial learning process. This can help to minimise the JSD difference as explained earlier.

Embedding function and recovery functions are auto encoding components whereas sequence generator and sequence discriminator are adversarial components. When both the components are trained jointly, timeGAN could effectively encode features and generate realistic sequences. The embedding network creates the latent space, within which the adversarial network functions. To ensure that both real and synthetic data share similar temporal dynamics, a supervised loss is applied, aligning the latent representations across time.

A. *Embedding and Recovery Functions* -

The embedding function takes the original data which consists of both static features as well as temporal features and convert it into a latent space. The latent space is a lower dimensional space where the data is represented in less complex manner. Recovery function helps the processed data in latent space transform back to the real form. The embedding and recovery functions needs to be autoregressive for parameterization.

B. *Sequence Generator and Discriminator*

Initially, Generator creates data using random vectors with the help of known patterns. Instead of providing the random generated data into the feature space, the Generator passes the output to Embedding Space. With the help of “Recurrent Networks” the Generator converts the random data into sequence of data for both static (data that doesn’t change over time) and temporal (data that change over time). The Discriminator plays the same role as usual to identify if the data is real or fake. The difference between Vanilla GAN and the updated timeGAN framework is that the Embedding Space helps Generator to learn more efficiently to generate realistic data especially for temporal type of data. The below figure illustrates the block diagram for components and loses (Yoon et. al., 2019).

Unsupervised Loss

Classifications

Reconstruction

Discriminate

Recovery

Supervised Loss

Latent Code

Embedding

Generate

Random Vectors

Real Sequences

Reconstruction Loss

C. *Amalgamation of Embedding Space and Generator*

The Generator passes the output data to Embedder, while the Recovery function feed the data into discriminator (Jeon et.al., 2022). The entire process produces three types of loses namely Unsupervised Loss, Supervised Loss and Reconstruction Loss.

**Reconstruction Loss** explains the recovery efficiency of the original data from the latent space. The embedding and recovery functions should be able to precisely reconstruct the original data. If the variation in reconstructed data is more then, the loss is high whereas if the variation is less then, the loss is low. It is represented as Lreconstruction. The equation is expressed as follows

**Unsupervised Loss** measures the efficiency of discriminator to determine real from fake data. Generator is exposed to two types of inputs. Firstly based on random vectors using autoregressive function and later improvising on the previous generated data. The objective of generator is to minimize this loss whereas generator tries to maximize it. The equation is expressed as follows

The feedback from the discriminator to generator (Adversarial Net) is not sufficient for the generator to capture all details, hence an additional loss is described for further disciplined learning. This is achieved by developing a closed-loop mode where it used real data for better generation. This lass is called as **Supervised Loss**. It measures the performance of generator’s prediction from actual data. The equation is expressed as follows

## 3.2. Methodology

With reference to the existing GAN architecture, the flow of the process to implement time GAN architecture on NASDAQ data is described as follows.

1. Data Preparation

* NASDAQ provides a dataset containing stock prices, dividends, and splits for 3,000 publicly traded U.S. companies. Before its acquisition on April 11, 2018, Quandl discontinued community support and updates for the data. This data can be downloaded at <https://data.nasdaq.com/tables/WIKIP/WIKI-PRICES/export>

This data is downloaded and imported using Pandas DataFrame. It is then processed it into a well-structured DataFrame with a multi-index of dates and stock tickers, before storing it in an HDF5 file for efficient storage and retrieval.

1. Setting up Parameters

* Once the file is imported, a directory is set up to save the experiment. The “experiment = 0” states that the initial experiment number is set as 0 and it will keep a track on different experiment data stored.
* The parameters for the experiment are set as follows

seq\_len = 24

n\_seq = 6

batch\_size = 128

The sequence length is taken as 24 because the seasonality of stock will be tracked 24 hrs. Since the thesis focus on analysing six stock companies, number of sequences are set as 6 and a common choice of batch size 128 is set for processing during training because of the size of the dataset. Six major blue chips stock companies namely DuPont (DD), Arconic (ARNC), Disney (DIS), General Electric (GE), IBM, and Coca-Cola (KO) will be analysed to compare the GAN performance (synthetic data vs real data). The selection of these 6 firms is random with an objective of ensuring that all companies are key players in the market.

1. Normalising data

Before combining and analysing the data, it needs to be standardised because the stock data has different ranges and characteristics. The stock signals (indicators) vary widely in value, hence, a feature scaling technique from Scikit-learn is used to pick out the most important features and reduce unnecessary noise. This is achieved using Scikit-learn MinMaxScaler function. It transforms the data such that all features are scaled into a specified range, between [0, 1] (Yadav et.al., 2023). Finally, we create overlapping groups of 24 data points, called rolling windows, to analyze trends smoothly over time. The function fit\_transform(df) helps the calculation of minimum and maximum from each column and transform it into the range 0 to 1. This data frame is named as “scaled data”.

1. Development of random series and real series data

Using TensorFlow, the scaled\_data of seq\_len 24 is sliced row by row for further processing. This sliced data is shuffled such that no pattern can be detected by the Generator model. This data is then grouped into the batch size of 128 as explained earlier.

The random series data is created using NumPy array function (yield np.random.uniform()). The stream of data is generated depending on seq\_len and n\_seq set earlier. Later, using TensorFlow, random\_series dataset is created and feed into Generator Function. The real data is denoted by X and random data is denoted by Z.

1. Describing GAN components used in TimeGAN

For time series data like stock prices, recurrent neural networks (RNNs) are well suited as they are designed to handle sequential information which is crucial for capturing temporal correlations. This property of RNN enables GANs to generate realistic synthetic time series (Dannels, 2023). Most popular RNN used in time series GANs are long short-term memory (LSTM) or gated recurring units (GRU). RNN is capable of handling variable length sequencing inputs. Although it is well established that LSTM unit is particularly learning long-term dependencies within sequences, GRU is more utilized for machine translation and handling long term dependencies. Chung et.al (2014) concludes that for fixed number of parameters, GRU could outperform LSTM in terms of convergence in CPU time as well as efficiency in parameter updates and generalization.

The model employs three GRU layers stacked on top of each other with each layer containing hidden units within each layer. This method is used across all components of the model, such as the generator, discriminator, embedder, and recovery functions. The timeGAN architecture in our case uses 2 major components. The Auto Encoding component encompass Embedding and Recovery as sub components whereas Adversarial Component encompasses Generator and Discriminator as sub components.

In the code a recursive function make\_rnn(n\_layers, hidden\_units, output\_units, name) is defined which returns a sequential data for number of GRU layers. The function make\_rnn() is used for creating 3 GRU layer RNN for all 4 sub components.

In Auto Encoding component, Encoder converts the data to latent space and Recovery reconstructs the latent space data into original form (refer point 1 in components of GAN in chapter 3). Similarly make\_rnn() is used for Adversarial components to create sequential data. As the Generator passes the data to Embedder, Recovery (Decoder) recovers it and converts it into Synthetic data.

Refer the image below for understanding the flow.

ADVERSARIAL COMPONENT

Adversarial Loss

Generator

Discriminator

Random Noise

Supervised Loss

Synthetic Data

Real Sequence

Embedding

Recovery

Recovery Loss

AUTO-ENCODING COMPONENT

1. GAN Training

Although GANs are widely used, training them remains difficult due to challenges like mode collapse and instability. To tackle these problems, various approaches have been developed to enhance GAN performance, including refining network architecture, adjusting loss functions, and applying regularization methods. However, few strategies prioritize optimizing GAN performance from the perspective of the optimizer, despite the significant impact different optimizers have on training. Most GAN models rely on a single optimizer, typically Adam, as the standard choice. While methods that incorporate multiple optimizers have shown better results, they are usually tailored to specific tasks (Zhang et al., 2024). Thus, before the pre training process, Adam() was used to optimize Autoencoder component. For the training process, different number of training steps ranging from 1000 to 20000 were considered to check the accuracy of real vs synthetic data. For convenience and to quickly identify the model with the highest performance, models were saved for every 1000 iterations during the 20000 training steps primary training.

Table 2. Parameters used for training

|  |  |  |
| --- | --- | --- |
| **SR. NO.** | **Parameters** | **Value** |
| 1 | Sequence Length | 24 |
| 2 | Number of stacked GRU layers | 3 |
| 3 | Batch Size | 128 |
| 4 | Optimizer | ADAM() |
| 5 | Training steps | 20000 |
| 6 | Number of Units | hidden\_dim |
| 7 | Loss Function  (Lreconstruction,LSupervised) | MSE |
| 8 | Loss Function (LUnsupervised) | BCE |

Another training step takes place during PCA and t-SNE run. Both are popular methods for dimensional reduction and it is implemented on real as well as synthetic data in order to compare the accuracy level and ROC Area Under Curve. The dataset for PCA and t-SNE is split in 80:20 ratio.

1. Calculation of Loses

T