# Introduction

### A primary challenge in modern AI and Machine Learning (ML) industry is scarcity of data. Datasets are often too limited in size, or the process of manually labeling large amounts of data becomes prohibitively expensive despite the ease of data collection. Data is essential for societal progress, and its widespread availability and analysis are crucial for informed decision-making. For example, application of ML in healthcare has the capability for predicting and managing epidemics, enhancing care quality, reducing healthcare costs, and advancing personalized treatments. However, these benefits remain largely untapped, because of restricted access to health data due to privacy concerns. A growing approach to addressing data availability challenges is the use of fully synthetic data (Dankar et al., 2021). The concept of synthetic data is one of the most modern and advanced method for addressing data challenges by either generating entirely artificial data from the ground up or applying sophisticated data manipulation techniques to create new and varied training examples. Synthetic data is increasingly important in privacy-sensitive fields like healthcare, finance, and social sciences, where sharing real data poses risks (Nikolenko, 2019). Unlike traditional anonymization techniques, which can still be vulnerable to re-identification attacks, synthetic data offers a stronger privacy safeguard (Jordan et al., 2022). Moreover, synthetic datasets can be tailored to address issues like data imbalance or scarcity, making them highly useful for training machine learning models.

### Related Work

### Research Question

### Scope and Limitation

### Report Outline

### Literature Review

The topic of Generative Adversarial Networks cannot be initiated without without the mentioning of Ian Goodfellow who not only introduced the GAN concept with his colleagues but has published several papers on GAN techniques and shortcomings. The concept of GAN is considered as a significant milestone in machine learning and generative modelling. Goodfellow et al. (2014) described the GAN framework as two player minimax game between two neural networks, the generator (G) and the discriminator (D). Both the neural network models compete against each other to prove each other wrong like a game theoretic set up. The constant competition between the generator and discriminator forces both models to become increasingly sophisticated, ultimately resulting in generated data that is virtually identical to real data. Prior to the arrival of GAN, traditional generative models such as Restricted Boltzmann Machines (RBMs), Variational Autoencoders, and Markov Chain Monte Carlo (MCMC) methods presented significant computational challenges. These models demanded complex mathematical approximations to estimate the probability of generating specific data points or employed complex sampling strategies. However, unlike Noise Contrastive Estimation (NCE), directly leverages the generative model to distinguish between generated samples and those from a fixed noise distribution. The issue with generative models were high computational cost (specially to train the model), scalability and incurring high loses during estimation.

Time series forecasting has always been a vital topic of interest for many years in research and application across different sectors. The ability to forecast upcoming trends in financial markets has been competitive for researchers and finance professionals for many years. This includes predicting the values of stocks, bonds, and various financial instruments, along with economic indicators such as inflation. These predictions are critical for formulating trading strategies, guiding decisions on the buying or selling of financial assets based on anticipated price changes. Traditionally, time series forecasting has focused on estimating a single future value. However, this study seeks to advance the field by utilizing Generative Adversarial Networks (GANs) to generate probabilistic forecasts. This approach enables the model to not only predict a single outcome but also to estimate the full probability distribution of potential future values of the target variable, such as stock prices. Traditional machine learning and time series models like ARIMA and LSTMs have shown some predictive power but failed to allow probabilistic forecasts. These methods often make rigid assumptions about data distribution, limiting their application in scenarios demanding uncertainty quantification, such as risk assessment, healthcare, and autonomous systems. GANs have higher Sharpe Ratios in comparison to the traditional methods. But one of the challenges in GAN is lack of efficient performance measures as its quite difficult to predict the distribution for generated data. Apart from MAE and RSME, profit and loss (PnL), ETF – excess, Sharpe Ratio etc. can also be included for better evaluation. In one of the extended GAN models named as Fin-GAN, introduces a novel economics-driven loss function in the GAN framework designed for financial applications. This function helps in shifting the generated data distributions to better align with real financial data, however, it introduces additional hyper parameters that needs to be fine-tuned during training. This increases the complexity due to difficulties in optimizing too many parameters (Vuletić et al., 2024).

Raghunathan (2021) compares the modern techniques of synthetization with the chemical synthesis process where human involvement mimics certain properties of natural materials. This process enables the broad use of key ingredients or products while conserving natural resources. In the similar way, using statistical methods, key insights from the real data can be used to develop synthesis (artificial data). The primary goal of synthetic data generation is to ensure that the original identity of the data is not exposed. Statistical Disclosure Control is a critical process for safeguarding respondent privacy when releasing public datasets. Nikolenko (2020) in his book regarding Synthetic Data for Deep Learning conducted a detailed survey and arrived on three primary reasons for using synthetic data.

1. **Training Models with Synthetic Data**: One approach involves training models exclusively on synthetic data with the goal of applying them to real-world scenarios. Alternatively, generative models can be used to refine synthetic data, enhancing its suitability for model training. The focus here is on improving the alignment between synthetic and real data.
2. **Augmenting Real Datasets with Synthetic Data:** Synthetic data can be combined with real datasets to create more robust hybrid datasets that enhance model performance. This is particularly useful for addressing data distribution gaps, where real data lacks sufficient representation in certain areas. Two common methods include:

* Generating synthetic data independently using techniques like CGI for computer vision tasks.
* Generating synthetic data from existing real data using generative models

1. **Addressing Privacy and Legal Constraints with Synthetic Data:** Synthetic data is increasingly valuable in contexts where privacy concerns or legal restrictions make the use of real data infeasible. This is particularly relevant in fields such as healthcare where synthetic data is used not only in imaging but also in areas like medical records. Financial firms such as banks and stock exchanges where direct applications are challenging, but privacy-focused synthetic data solutions are emerging. Lastly, for sensitive applications that require stringent privacy protections, such as those guided by differential privacy frameworks, recent advances have focused on generating synthetic data with built-in privacy guarantees.

The application of synthetic data for data augmentation is mainly required during the training of deep neural network where data labelling is essential. Generating labelled data is an expensive affair, hence data synthesis makes it cost efficient. The main concept behind using synthetic data for data augmentation is to enhance generalizability. A model’s objective is not only to excel on its training data but also to perform well on new, unseen data. B leveraging the available training data to train a generative model, such as a GAN, one can generate "realistic" samples. Ideally, these generated samples will be diverse enough from the original training data to serve as valuable new data points, improving the model’s overall performance during training. GANs were developed with the ability to be both parallel and multi-modal, meaning they can generate multiple acceptable outputs for a given input. GANs consist of two competing networks: the generator and the discriminator. These networks are trained in an adversarial manner. The generator produces synthetic outputs, which are then evaluated by the discriminator alongside real data. The discriminator’s role is to distinguish between genuine and "fake" outputs. The ultimate objective is to reach a state of equilibrium, where the generated samples closely match the distribution of the real data. However, as per the voice of business, its still debatable whether synthetic data can wholly replace real data. Although differential privacy is an appealing theoretical concept, industries find it challenging to fully trust it without empirical evidence demonstrating its effectiveness in real-world scenarios. But with current challenges on data availability, synthetic data is the way to go (Jordan et al., 2022).

### Generative Adversarial Network for generating synthetic time series data and evaluation

### Development of Synthetic Data

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.

### 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, Conditional GAN (CGAN) 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

### 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.

### 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.

### 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.

### 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”.

### 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.

### 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 GAN overview referred in the thesis.

ADVERSARIAL COMPONENT

Adversarial Loss

Generator

Discriminator

Random Noise

Supervised Loss

Synthetic Data

Real Sequence

Embedding

Recovery

Recovery Loss

AUTO-ENCODING COMPONENT

### 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) | 24 |
| 7 | Loss Function for embedding\_loss and generator\_loss\_supervised  (Lreconstruction,LSupervised) | MSE |
| 8 | Loss Function for generator\_loss\_unsupervised and discriminator\_loss  (LUnsupervised) | BCE |

**Algorithm 1** Training Loop for Autoencoder and Suprvisor

1: **Initiate Training Loop:** for step in tqdm(range(train\_steps)): X\_ =next(real\_series\_iter)

*/\** setting up iteration for real data\*/

2: **Traing Autoencoder:**  step\_e\_loss\_t0 = train\_autoencoder\_init(X\_)

3: **Tensorflow summary for output** : tf.summary.scalar('Loss Autoencoder Init', step\_e\_loss\_t0, step=step)

This code enables the autoencoder to train the real data (X) for calculating e\_loss\_t0 and g\_loss\_s. The methodology on calculating loses will be explained in further sections.

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.

### Calculation of Loses

The training of the Autoencoder and Adversarial components includes loss calculation. The qualitative evaluation techniques for calculating loses includes the use of metrics associated with statistical measures used specifically for time series such as Pearson Correlation Coefficient (PCC) Mean Squared Error (MSE and RMSE), Mean Relative Error (MRE), Mean Absolute Error (MAE) and Binary Cross Entropy (BCE). These metrics are widely used for evaluating time series data and are considered appropriate for assessing GAN performance, as they reflect the consistency between the training data and the synthetic data generated (Brophy et al., 2022). TimeGAN utilizes the mean squared error (MSE) as the generative loss function to measure the distance between real and fake discriminator scores. It is also combined with other loss functions such as adversarial loss. Binary Cross Entropy determines the probability of correct classification of real vs fake data. Hence it is used for calculating the Discriminator loss function. Of all the evaluation metrics mentioned, training a model solely with BCE loss will lead the model to produce symmetrical distributions. This is because BCE loss focuses on minimizing the error between the predicted probability and the true label, without explicitly considering the shape of the distribution (Vuletic et al., 2023). The role of the discriminator function is to maximize the BCE loss value whereas the Generator will try to minimize the MSE value. Now, because the MSE loss function directly compares the generated output to the desired target, the generator can learn the target distribution very effectively, even if the discriminator is not performing optimally (Xiao, 2023). Thus, MSE and BCE are defined before initiating the loss calculation. In this thesis, TensorFlow tf.function() is used for training Autoencoder, Supervisor, Generator, Embedder and Discriminator. Go through the list of loses defined below.

**A. Embedding Loss -** The real data (X) is passed through auto encoder to extract reconstructed data X\_tilde. MSE calculated between *X* and *X\_tilde* will provide us the embedding\_loss (*e\_loss\_0*) value.

**B. Generator Supervised Loss** - H is the hidden representation of real data when it’s passed through the Embedder. Training the Supervisor(H) will provide the values for *H\_Hat\_supervised*. The MSE calculated between *H* and H\_Hat\_supervised will provide the *g\_loss\_s* value. The function *get\_generator\_moment\_loss* calculates the total losses generated through mean and variance of *g\_loss*. Through Tensorflow function tf.nn.moments, mean and variance can be calculated of the real data and its named as *y\_true* and *y\_pred*. The *g\_loss\_mean* is the difference *y\_true* and *y\_pred* whereas *g\_loss\_var* is the standard deviation of their distributions.

**C. Generator Unsupervised Loss –** The *y\_fake* data is created using adversarial\_supervised (z) function, where z is the random noise and *y\_real* data is created from the recovered data after passing through embedder (H). The BCE calculated between *y\_fake* and *y\_real* provides the *generator\_loss\_unsupervised (g\_loss\_u)* value. The Unsupervised loss has another component denoted as *generator\_loss\_unsupervised\_e* which is calculated using function *adversarial\_emb()*. Note that *adversarial\_supervised()* and *adversarial\_emb()* functions are created using the algorithm as mentioned below

**Algorithm 2: Defining the adversarial\_supervised function**

**Input -** E\_hat = generator(Z)

H\_hat = supervisor(E\_hat)

Y\_fake = discriminator(H\_hat)

Y\_fake\_e = discriminator(E\_hat)

*/\** flow of data from Generator, Supervisor and Discriminator \*/

**Model –**

adversarial\_supervised =Model(inputs=Z,outputs=Y\_fake,name='AdversarialNetSupervised')

adversarial\_emb = Model(inputs=Z, outputs=Y\_fake\_e, name='AdversarialNet')

/\* The input to this model is the random noise Z, and the output is the discriminator's decision on whether the generated data is real or fake (Y\_fake). This structure allows for backpropagation of gradients during for GAN training step. \*/

**D. Generator Loss –** The total generated loss function is calculated from the equation that consists of Generator Supervised Loss (g\_loss\_s), Generator Unsupervised Loss (g\_loss\_u) and Generator Moment Loss (get\_generator\_moment\_loss). Mathematically it can be represented as

**E. Discriminator Loss –** Discriminator loss denoted as *discriminator\_loss (d\_loss)* is divided into 2 components namely *discriminator\_loss\_real* and *discriminator\_fake.* As explained in the previous sections, BCE is used to calculate the discriminator loss functions because it determines the probability of correct classification of real vs fake data. BCE loss focuses on minimizing the error between the *y\_pred* and the *y\_true*.

The calculation for discrimination loss can be mathematically represented as

Where,

* Lreal​ is the loss for correctly classifying real data.
* Lfake​ is the loss for correctly classifying fake data.
* Lfake\_e is the loss for classifying fake embeddings.
* Gamma is a hyperparameter.

## Chapter 4

### Performance Evaluation and Analysis of Generative Adversarial Networks to create synthetic dataset



### Environment Setup

### Hardware

Concerning the experimental phase, all of the GANs models were developed and trained using Tensor Flow and Keras. The training process was carried out on a computer with a CPU Intel Core i7 with 8GB of RAM. The process of detecting NASDAQ stock price prediction by generating datasets from Generative Adversarial Nets requires higher computational capabilities. We have Jupyter Lab, a free to use public cloud platform based on Jupyter Notebooks for training, validation, and testing. GAN utilizes NVIDIA GPU for training, hence the system needs to install GPU version of TensorFlow or Pytorch. deep learning models. Here are the recommended hardware requirements:

CPU

• Type: Multi-core processor, preferably with high clock speeds.

• Recommended: Intel Core i5, i7 or i9, AMD Ryzen 7 or 9, or equivalent.

GPU

• Type: High-end CUDA-compatible GPU.

• Recommended: NVIDIA RTX 30 series (e.g., RTX 3090, RTX 3080) or NVIDIA A100 for more extensive tasks.

RAM

• Minimum: 8 GB

• Recommended: 16 GB or more

### Software Requirements

Many packages and libraries are installed for the implementation of timeGAN. As explained in the chapter 3, for the development of Generative Adversarial Networks, we use Pytorch and Tensorflow packages. The list of software packages are as follows

1. **Python (Anaconda)** – Anaconda Navigator is a graphical user interface (GUI) that enables you to work with packages and environments without needing to type conda commands in a terminal window. For the ease of accessing various python notebooks without setting up environment, Firstly ANACONDA navigator was downloaded for implementing GAN because it is inclusive of all data science tools. Within ANACONDA navigator Jupyter Lab (version 4.0.11) was used for downloading and installing all packages, datasets and directory set ups.
2. **Pandas and Pandas- datareader** – Pandas.datareader is installed for reading and writing data from in-memory financial data structures and other CSV formats. (Pandas, 2024). Pandas data reader is handy while working with time series datasets due to its functionality with date range generation.
3. **Scikit Learn -** The Sklearns-Feature selector function (Scikit-Learn, 2007–2024) is utilized to filter out unnecessary noise by selecting significant features. NumPy a subset of Scikit-Learn is used for subsequently normalized using feature scaling, adjusting all values within the range of [0, 1], which ensures consistency across the dataset. This process standardizes all features to have the same format and range. Additionally, the Scikit-MinMaxScaler function was applied, and rolling windows with overlapping sequences of 24 data points were generated.
4. **Tensorflow and Keras -**  Numerous packages have been imported for implementing tensorflow.keras functions such as Sequential Models, GRU, Dense, RNN, Adam(), BCE, MSE, TensorBoard etc. The entire timeGAN coding and algorithm is heavily relied on Tensor Flow libraries.
5. **Matplotlib and Seaborn -**  Matplotlib and Seavorn is used for all Machine Learning projects for visualization. In this thesis, its used for creating cluster maps and line plots.
6. **Tqdm –** tqdm which is inspired from the Arabic word “taqadum” (means progress) is used for optimizing the training loops within the TimeGAN (tqdm, 2015-2022). It is used in generating training loops for autoencoder.
7. **Pydot and graphiz -** Pydot is a library that is used to plot the flow of GAN components during the training. Unlike other software, Pydot and graphiz needs to be installed on the OS terminal rather than Jupyter Notebooks. For this thesis, Pydot has been selected as Markdowns because the installation was unsuccessful.

### Dataset

This data feed offers stock prices, dividends and splits for 3000 US publicly-traded companies. The entire WIKI data feed is stored as a single table for faster and easy retrieval. Tables are a collection of data structured as 14 columns. NASDAQ makes available a dataset with stock prices, dividends and splits for 3000 US publicly-traded companies. Prior to its acquisition (April 11, 2018), Quandl announced the end of community support (updates). The historical data are useful as a first step towards demonstrating the application of the machine learning solutions, just ensure we design and test your own algorithms using current, professional-grade data. For this dissertation, 2 primary datasets are used. One dataset (Wiki Prices) consists of stock prices of all companies from 1962 to 2018 whereas another datasets(Wiki Stocks) consists of codes and names of 3199 listed companies. Of all the available columns, below is the description of a few vital columns.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr.No** | **Field** | **Quotes / Trades** | **Type** | **No Value** | **Description** |
| 1 | ticker |  | String | Never | Ticker Symbol |
| 2 | date |  | YYYYMMDD | Never | Trade Date |
| 3 | open | T | Number | Never | Price of the first trade |
| 4 | high | T | Number | Never | Price of highest Trade |
| 5 | low | T | Number | Never | Price of lowest Trade |
| 6 | close | T |  |  | Price of the last trade |
| 7 | volume | T |  |  | Total number of shares traded |
| 8 | ex-dividend |  | String | Never | Number of days for next divident |
| 9 | split\_ratio |  | String | Never | Ratio of number of shares before split to shares after split |

The dataset used for this project is taken from the publicly available GitHub repository given in (Jansen et al. 2020). The source datasets is directly downloaded from NASDAQ.

### Performance Evaluation of TimeGAN

GANs are a powerful deep generative model trained with an adversarial procedure. Similar to SMOTE, GANs have undergone several modifications since they were first proposed to solve several different problems in different domains

### Cluster Map

Before the start of generating synthetic data, it is important to view the trend and patterns described in the dataset. While setting up the experimental path during parameters set up, 6 number of sequences were mentioned based on the tickers. Seaborn cluster map is a clustered heatmap of the correlation matrix with annotated correlation coefficients among all the 6 selected tickers, providing a clear visual representation of the relationships between different features in our dataset. From the cluster map, correlation among the tickers can be interpreted. KO (Coca-Cola), DD (DuPont), and DIS (Disney) are closely clustered, indicating strong positive correlations among them. From the colour gradient, the darker the shade of blue, more the correlation whereas red shade indicates lesser correlation. IBM and KO have a strong positive correlation (0.89), indicated by the dark blue color. ARNC (Arconic) shows a negative correlation with several other stocks, such as -0.69 with GE and -0.66 with IBM, indicated by the red color gradient.

A screenshot of a computer screen

Description automatically generated

### Trend Plots

The below figure represents trend plots for the 6 selected tickers namely DuPont (DD), Arconic (ARNC), Disney (DIS), General Electric (GE), IBM, and Coca-Cola (KO). From the trend graph, its noticeable that IBM, DuPont, Coca-Cola and Disney stock prices were less fluctuating upto 2010 and all the 3 stocks surged its way towards upward trend post 2010. On the other hand, GE and Arconic stocks followed a downward trend with some recovery in between after 2010. Disney and Coca-Cola is trending in similar directions, suggesting a potential correlation between the two companies' performance. One of the objectives for plotting trends were to visualize the data distribution for the selected stock companies.

A graph of different trends

Description automatically generated with medium confidence

### Auto Encoder Summary

An Autoencoder model is primarily used for data compression and dimensionality reduction. As the name suggests, the auto encoder helps in embedding and recovery of data. The model consists of two main parts: the Encoder (Embedder) and the Decoder (Recovery).

* Embedder: Compresses the input data into a lower-dimensional representation (latent space).
* Recovery: Reconstructs the original data from the lower-dimensional representation.

There are different types of autoencoder depending on the usages such as Convolutional Autoencoder (CAE) is used for decoding image data. In our case, the data generation is through RNNs (GRU, 3 layers), Variational Recurrent Autoencoder (VRAE) is ideal for summarization. VRAE is widely used for time series data. VRAE is a model combining the RNN with the variational method. Autoencoders are trained to minimize the difference between the input and the reconstructed output, typically using a loss function like mean squared error (MSE) (Jeon et al., 2023).

If the data is normalized then the Autoencoders can be trained to learn typical patterns (B Zong, et al. 2018). In this case, we have followed the same mechanism by setting up a random series using RNN that continues to call the random number generator as long as necessary and produces the desired batch size. Once the parameters for Generator and Discriminator are set up using random series and real data, the training loop are set for 20000 steps. The auto encoder provides the results as follow

A table with numbers and symbols

Description automatically generated with medium confidence

**Model Summary**

1. Real Data (Input layer) –

Output Shape - The normalized real data has the sequence length 24 with 6 features (tickers). None indicates a flexible batch size. It is none because the value is not mentioned.

Param – It indicates parameters which is 0 for input data.

1. Embedder (Sequential)

Output Shape – Embedder acts as an encoder to convert input sequence (X) of 24,6 into a latent space data (H) of size 24,24

Param – The embedder layer has 10,104 trainable parameters for encoding.

1. Recovery (Sequential)

Output Shape – Recovery layer acts as a decoder to reconstruct the latent space data (H) of 24,24 into the original shape of size 24,6

Param – The recovery layer has 10,950 trainable parameters for decoding the latent data.

The sum of all trainable parameters is 21,054. This indicates that these parameters will be available for backpropagation. Backpropagation is a method to optimize the parameters of RNN by calculating how much each parameter (trainable) contributes to the overall error and adjusting it accordingly (Bengio et al., 2017).

### Adversarial Net Supervised Summary

To recap the loses section in chapter 3, Adam() was used to optimise the Autoencoder and Supervisor before applying tensor flow function creating embedding loss and supervised loss. Now, we generate a model named “Adversarial Net Supervised” that will help a generator to train using random data.

A screenshot of a computer

Description automatically generated

**Model Summary**

The components used for developing this model are Generator, Supervisor and Discriminator. The flow of data can be illustrated as

Random Data 🡪 Generator 🡪 Supervisor 🡪 Discriminator

The Adversarial Net Supervised takes the input data Z (random data) and create a discriminator’s output data. Let’s understand the outputs for each layer.

1. Random Data (Input layer) –

Output Shape - The normalized real data has the sequence length 24 with 6 features (tickers). None indicates a flexible batch size. It is none because the value is not mentioned.

Param – It indicates parameters which is 0 for input data.

1. Generator (Sequential)

Output Shape – The generator transforms the input sequence (E\_hat = generator(Z)) of shape 24,6 into a latent space (higher dimension) data of shape 24,24. Note that Z is the random series data and E\_hat is the output indicator of generator.

Param – The generator layer has 10,104 trainable parameters including weight and bias of RNN.

1. Supervisor (Sequential)

Output Shape – The supervisor layer enhances the latent space data (E\_hat) of shape 24,24 process it further (H\_hat = supervisor (E\_hat))

Param – The supervisor layer has 7,800 trainable parameters including weight and bias of RNN.

1. Discriminator (Sequential)

Output Shape – The discriminator takes the processed data by the supervisor (H\_hat) of shape 24,24 and predicts if the sequence data is real or fake. The output shape of discriminator is 24,1 (Y\_fake = discriminator(H\_hat)). Y\_fake is the output result provided by the discriminator for the enhanced data from supervisor layer.

Param – The discriminator layer has 10,825 trainable parameters including weight and bias of RNN for identifying fake vs real data.

The sum of all trainable parameters is 28,729. This indicates that these parameters will be available for backpropagation.

### Adversarial Net Summary

The neural network model named “Adversarial Net” is a simpler version of the earlier model excluding the supervisor layer for training random data.

A screenshot of a computer

Description automatically generated

**Model Summary**

The components used for developing this model are Generator and Discriminator. The flow of data can be illustrated as

Random Data 🡪 Generator 🡪 Discriminator

The Adversarial Net Supervised takes the input data Z (random data) and create a discriminator’s output data. Let’s understand the outputs for each layer.

1. Random Data (Input layer) –

Output Shape - The normalized real data has the sequence length 24 with 6 features (tickers).

Param – It indicates parameters which is 0 for input data.

1. Generator (Sequential)

Output Shape – The generator transforms the input sequence of shape 24,6 into a latent space data of size 24,24.

Param – The generator layer has 10,104 trainable parameters including weight and bias of RNN.

1. Discriminator (Sequential)

Output Shape – The discriminator classifies the higher dimensional latent data of shape 24,24 and predicts if the sequence data is real or fake. The output shape of discriminator is 24,1 (Y\_fake\_e = discriminator(E\_hat)). Y\_fake\_e is the output result provided by the discriminator from the generator layer.

Param – The discriminator layer has 10,825 trainable parameters including weight and bias of RNN for identifying fake vs real data.

The sum of all trainable parameters is 28,729. This indicates that these parameters will be available for backpropagation.

### Synthetic Data Summary

The Synthetic Data model in a time series GAN is a complex tool that creates high-quality fake time series data. It takes random noise and passes it through several steps to turn it into data that looks like real time series data (Goodfellow et al., 2015). This summary model develops the synthetic data from the generator that has reconstructed itself to the original form through Recovery function.

A table with numbers and letters

Description automatically generated

**Model Summary**

The components used for developing this model are Generator, Supervisor and Recovery layers. The flow of data can be illustrated as

Random Data 🡪 Generator 🡪 Supervisor 🡪 Recovery

The synthetic data model takes the input data Z (random data) and create a discriminator’s output data. Let’s understand the outputs for each layer.

1. Random Data (Input layer) –

Output Shape - The normalized real data has the sequence length 24 with 6 features (tickers).

Param – It indicates parameters which is 0 for input data.

1. Generator (Sequential)

Output Shape – The generator transforms the input sequence of shape 24,6 into a latent space data of size 24,24.

Param – The generator layer has 10,104 trainable parameters including weight and bias of RNN.

1. Supervisor (Sequential)

Output Shape – The supervisor layer enhances the latent space data of shape 24,24 and process it further.

Param – The supervisor layer has 7,800 trainable parameters including weight and bias of RNN.

1. Recovery (Sequential)

Output Shape – Recovery layer acts as a decoder to reconstruct the processed supervisor data (H\_hat) of 24,24 into the original shape of size 24,6

Param – The recovery layer has 10,950 trainable parameters for decoding the latent data.

The sum of all trainable parameters is 28,854. This indicates that these parameters will be available for backpropagation.

### Discriminator Summary

The discriminator model helps to distinguish between real time series data and data generated by the generator. By providing accurate feedback to the generator, the discriminator helps improve the quality of the generated time series data, which is critical for tasks like anomaly detection, forecasting, or any application requiring realistic synthetic data.

A screenshot of a computer

Description automatically generated

The components used for developing the discriminator model are Real Data, Embedder and Discriminator layer. The flow of data can be illustrated as

Real Data 🡪 Embedder 🡪 Discriminator

The real data has a shape of 24, 6 as defined during parameter settings. The layer likely processes the input data and projects it into a higher-dimensional space (24 features). It contains 10,104 parameters, suggesting a relatively complex embedding process.

Discriminator is the core part of the model, which takes the embedded data and outputs a single value (likely a probability) indicating whether the input is real or fake. It contains 10,825 parameters.

The total parameters (20,929) indicate the model's complexity, with all parameters being trainable, this ensures that the model can adapt and learn effectively from the data. The discriminator model is essential during the entire training process, as it enforces the accuracy standards for the generator's output, thereby improving the model's performance in detecting anomalies in multivariate time series data (Yongshan, et al, 2024).

### Training Logs from GAN

After setting the 20000 training steps and developing 4 summary models as explained above, Adaptive Moment Estimation (ADAM()) is used to optimise generator , discriminator and embedder data.

Post optimization, following loses are calculated during the training of each model as listed below

* Embedding Loss (e\_loss\_t0)
* Generator Supervised Loss (g\_loss\_s)
* Generator Unsupervised Loss (g\_loss\_u)
* Generator Variance Loss (g\_loss\_v)
* Discriminator Loss (d\_loss)

The training progress of the Generative Adversarial Network (GAN) model shows the values of different loss functions for 20000 training iterations.

A table of numbers with a loss

Description automatically generated with medium confidence

**Embedding Loss –**

* **Initial (0): 0.0436**
* **After 20000 steps: 0.0027**

The embedding loss (e\_loss\_t0) decreases, indicating that the model's embedding layer is learning to represent the data more accurately. This reduction signifies improved performance in capturing the underlying structure of the data at the initial time step.

**Supervised Generator Loss –**

* **Initial (0): 0.0003**
* **After 20000 steps: 0.0001**

The supervised generator loss (g\_loss\_s) remains very low and constant, indicating stable performance in generating data that adheres to the supervised signal. This stability is crucial for maintaining the generator's focus on producing high-quality synthetic data .

**Variance Loss -**

* **Initial (0): 0.4279**
* **After 20000 steps: 0.0233**

The variance loss (g\_loss\_v) decreases significantly, which suggests that the synthetic data's variance is becoming closer to the real data's variance. This is a positive sign that the generator is not only producing realistic samples but also maintaining the statistical properties of the real data.

**Unsupervised Generator Loss –**

* **Initial (0): 0.6034**
* **After 20000 steps: 1.5382**

The unsupervised generator loss (g\_loss\_u) increases, which can indicate that the Generator is improving in producing more diverse and realistic samples that the discriminator finds harder to classify correctly. This increase shows that the generator's output is becoming more sophisticated.

**Discriminator Loss –**

* **Initial (0)**: 2.162
* **After 20000 steps**: 1.3487

The discriminator's loss (d\_loss) decreases over time, indicating that it is getting better at distinguishing between real and synthetic data. Lower discriminator loss suggests that the Generator is producing more realistic samples, making the discriminator's task more challenging . The methodology of calculating loses are covered in section 3.4.

### Performance Analysis of GAN

After the loses are calculated, the generated\_data was created by appending the synthetic and random series data. The shape of the generated dataset is (4352, 24, 6) where 4352 represents the samples, 24 represents number of time steps with 6 features per time step. The synthetic data created from the synthethic\_data summary model is saved in the assigned directory as “syntethic\_data.npy”. When the syntethic dataset is plotted against the real data sample, it resulted into below figure.

A group of graphs showing different types of data

Description automatically generated

From the visual comparison of Real data vs Synthetic data, the graph indicates that although the synthetic data does not fully match with the real data but the Generator was successful in learning the trends especially for IBM and DuPont. The current generated synthetic data in TimeGAN is somewhat capable of learning to create useful data representations, encode specific characteristics, and process data over time. If TimeGAN is compared to MTS-TGAN , the performance can be further improved due to its higher ability and efficiency to decode data and handle multi-dimensional data (Yadav et al., 2023).

The synthetic data is getting the trend of the real data but it was not highly successful in capturing the trends accurately for all 6 companies. Thus a better attempt can be made to reduce the visible differences from the line graphs of synthetic vs real data for all 6 listed companies.

### Qualitative comparison of real and synthetic data distribution

If the synthetic and real datasets are merged, then t-SNE can be applied to reduce its dimensionality to 2D for visualization or further analysis

A comparison of a data distribution

Description automatically generated with medium confidence

The image presents a comparison of data point distribution for real and synthetic data distributions using two dimensionality reduction techniques: Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE). The data distribution in orange are the synthetic data whereas blue represents the real data.

The PCA plot shows that the real and synthetic data have similar overall distribution shapes. This indicates that the synthetic data generator has successfully captured some of the real data's underlying structure. Although there are some overlap between the real and synthetic data points. This overlap suggests that the synthetic data does not perfectly capture all the variations found in the real data. But density wise, synthetic data points have a similar density to the real data points in some areas. This similarity implies that the generator has effectively captured certain local characteristics of the real data.

The t-SNE plot shows more detailed patterns in the data. The real and synthetic data have similar clustering patterns, but there are also clear differences. Some clusters have a lot of overlap between real and synthetic data points, which indicates that the generator has successfully captured some local structures. However, in other areas, the synthetic data forms separate clusters from the real data, indicating the generator may have missed some detailed aspects of the original data. Based on both the graphs, it appears that the synthetic data generator has achieved a reasonable level of success in capturing the overall distribution and some local structures of the real data but there are still areas where the synthetic data differs significantly from the real data, indicating potential limitations of the generator.

### Time series classification performance

A white box with black text and blue text

Description automatically generated

We run a time series classifier model using RNN layers namely GRU and OUT. Here the model has total of 259 parameters. All 259 parameters are trainable, meaning they will be adjusted during the training process.

The accuracy of the time series classifier came out to be 59.47%.

A graph of a performance

Description automatically generated with medium confidence

The Accuracy and ROC curve for ts\_classifier.evaluate() indicates that the model performs sufficiently well on training data. For Accuracy, the Area Under Curve for training data is over 90%, whereas AUC for test data stabalises around 60% to 65%. This discrepancy indicates that the model might be overfitting to the training data.

Similarly the Area Under Curve for ROC indicates that the training AUC increases quickly, approaching 1.0 within the first 50 epochs and keep maintaining the same value with minor fluctuations. The test AUC starts low (around 0.2) and gradually improves, stabilizing around 0.65 with minor fluctuation.

Time series prediction performance on train vs test data also indicates that the Mean Absolute Error for train data is quite lesser than the test data.

A graph of different colored lines

Description automatically generated with medium confidence

### Conclusion and Future Works

### Conclusions

### Future Works