## Chapter 4

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



### Environment Setup

### Hardware

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

For the development of Generative Adversarial Networks, we use Pytorch and Tensorflow. PyTorch is a well-known python based deep learning framework framework utilized for training and building neural networks (Paszke et al. 2019). PyTorch is widely used for Computer Vision (CV) and Natural Language Processing (NLP) applications as it allows the usage of GPUs for training neural networks which enormously reduces the training time. Scikit-learn (Pedregosa et al. 2018), math (van Rossum and Drake 2011), seaborn (Bisong 2019b), and matplotlib(Bisong 2019b)are the other libraries used for the creation of confusion matrix, evaluation metrics calculations, and plotting various graphs.

### 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 |

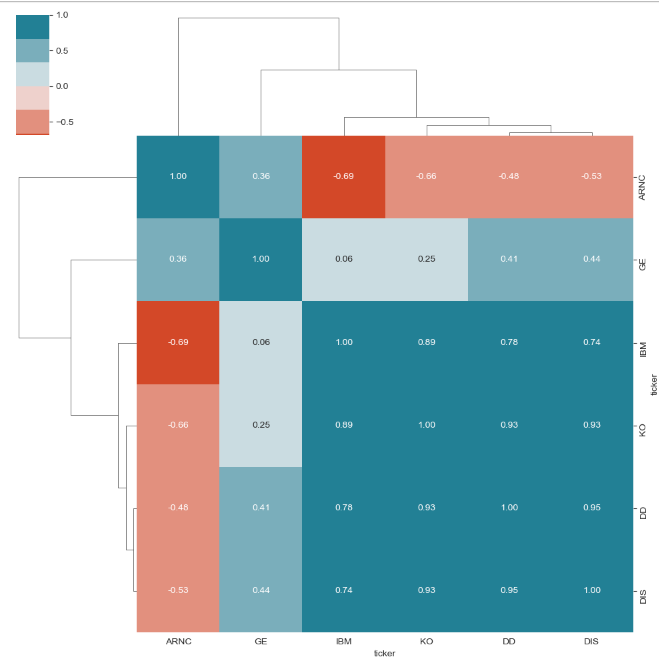
The dataset used for this project is taken from the publicly available GitHub repository given in (Jansen et al. 2020). The source datasets is directly downloaded from NASDAQ.

## Performance Metrics for Classification

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

Seaborn cluster map is a clustered heatmap of the correlation matrix with annotated correlation coefficients, providing a clear visual representation of the relationships between different features in our dataset. It is a powerful tool for visualizing hierarchical clustering of data matrices. It combines heatmaps and dendrograms to show both the data values and the clustering relationships.



### Auto Encoder Summary

An Autoencoder model was implemented to learn efficient representations of data, typically for dimensionality reduction or feature learning. The model consists of two main parts: the Encoder (Embedder) and the Decoder (Recovery).

* Encoder: Compresses the input data into a lower-dimensional representation.
* Decoder: Reconstructs the original data from the lower-dimensional representation.

Autoencoders are trained to minimize the difference between the input and the reconstructed output, typically using a loss function like mean squared error (MSE).

It can reduce the dimensionality of data while preserving its essential structure and features. This is similar to Principal Component Analysis (PCA), but autoencoders can capture more complex, non-linear relationships (Hilton et al. 2008). In this paper, we are normalizing the data so that the Autoencoders can be trained to learn typical patterns (B Zong, et al. 2018). By setting up a random series generator to feed a tf.data.Dataset 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

The provided autoencoder model was executed using Recurrent Neural Networks (RNNs) . RNNs are particularly suited for sequence data due to their ability to capture temporal dependencies (Sutskever et al., 2014). the Embedder and Recovery networks using typical RNN layers, such as LSTM or GRU, in a Keras-based autoencoder model.

**Parameter Explanation**

 LSTM Layer Parameters: The parameters for an LSTM layer are calculated as:

Params = 4 X (units X (units + input\_dim +1)

where units is the number of LSTM units and input\_dim is the number of input features. The factor of 4 comes from the gates in an LSTM (input, forget, cell, and output gates) .

**Dense Layer Parameters**: The parameters for a Dense layer are calculated as:

Params = (input\_dim +1) x units

where input\_dim is the number of input features and units is the number of output units (Goodfellow et al., 2016).

With relatively small number of parameters (21,054) suggests that the model is lightweight and can be trained relatively quickly.

### Adversarial Net Supervised Summary

Adversarial Net Supervised model integrates a Generator, a Supervisor, and a Discriminator into a single adversarial setup with supervised constraints. It refers to a model architecture that combines adversarial learning with supervised learning. This hybrid approach is often used in scenarios where one needs to leverage both the generative adversarial network (GAN) framework and additional supervised signals to improve the performance and capabilities of the model. Adversarial learning typically involves Generator for creating synthetic data and Discriminator for evaluating the authenticity of realistic the synthetic data (Radford et al., 2015).

A screenshot of a computer

Description automatically generated

As per the results shown above, the generator has 10,104 trainable parameters, specifying that the model has a moderately complex structure. The supervisor network, with 7,800 parameters, likely adds more refinement to the generated data, ensuring the generated sequences are coherent and realistic. The discriminator, with 10,825 parameters, evaluates the realism of each generated sequence. The output shape suggests it might be providing a real/fake score for each time step.

With a total of 28,729 parameters, the model has a moderate capacity for learning from data. This capacity allows it to handle complex tasks such as data generation and evaluation, but the specific architecture (e.g., number of units in layers) will determine how effectively it can learn and generalize.

### Synthetic Data Summary

The model summary for "SyntheticData" in a time series GAN (Generative Adversarial Network) model provides an overview of the layers and parameters involved in generating synthetic time series data. The "SyntheticData" 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).

A table with numbers and letters

Description automatically generated

The input shape (None, 24, 6) indicates sequences with 24 time steps and 6 features per time step. The Generator (sequential) layer processes the random noise input to create an initial synthetic representation. It increases the feature space from 6 features per time step to 24 features per time step, resulting in an output shape of (None, 24,24). The number of parameters (10,104) reflects the complexity of the transformations performed by the generator. The Supervisor layer further refines the synthetic representation created by the generator. It maintains the feature space at 24 features per time step, enhancing the quality of the synthetic data. The number of parameters (7,800) indicates the additional transformations applied to the synthetic data.

With a total of 28,854 parameters, the model is more complex than the discriminator but still relatively lightweight, balancing computational efficiency with the ability to generate realistic data.

### 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 input layer for the data has a shape of (None, 24, 6). This means each data point is a sequence of 24 time steps, and each time step has 6 features. 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, which ensures that the model can adapt and learn effectively from the data. The discriminator module 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, 4 summary models were produced as explained above. Adaptive Moment Estimation (ADAM()) was used to optimise generator , discriminator and embedder data.

After the optimisation of all 3 models, they were trained for calculating computing losses from the supervised adversary, embedding adversary, supervised data reconstruction, and moment matching.

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

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 .

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 .

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.

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 .

## Performance Analysis of GAN

Now that the loses are calculated, a 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. When the generated dataset was plotted, it resulted into below figure.

A group of graphs showing different types of data

Description automatically generated

1. **Qualitative comparison of real and synthetic data distribution**

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.

By preparing a dataset through merging real and synthetic data, then applies t-SNE 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 real and synthetic data distributions using two dimensionality reduction techniques: Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE).

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, meaning the generator has successfully captured some local structures. But 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.

1. **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