Comparison of conventional deep learning models and Generative Adversarial Network for stock price prediction

QF634 – Applied Quantitative Research Methods

G2 : Group D

Group Members:

Balina Kirti Kumar

Kendrick Winata

Najwa Jia Hui Rujok

Saran Somboonsiripok

MSc in Quantitative Finance

Academic Year 2023 – 2024

Table of Contents

		Page
1.)	Abstract	1
2.)	Introduction	1
3.)	Literature Review	1
4.)	Theoretical Background	2
	4.1) Long Short-Term Memory (LSTM)	2
	4.2) Gated Recurrent Unit (GRU)	3
	4.3) Convolutional neural network (CNN)	3
	4.4) Generative Adversarial Network (GAN)	4
5.)	Datasets and Features	5
	5.1) Datasets	5
	5.2) Features	5
6.)	Methodology	6
	6.1) Data Preprocessing	6
	6.2) Generator and Discriminator for GAN framework	7
	6.3) GAN Framework Structure	8
	6.4) LSTM Model	10
	6.5) GRU Model	10
7.)	Experimental Results	10
	7.1) Performance measurements	10
	7.2) Experimental Results	11

List of Figures

		Page
Figure 1	Data preprocessing procedures	6
Figure 2	Architecture of GAN framework	9
Figure 3	Observable patterns in the loss functions of GAN, LSTM and GRU	12
Figure 4	Prediction results of AAPL by GAN framework	12
Figure 5	Prediction results of AAPL by GRU	12
Figure 6	Prediction results of AAPL by LSTM	13
Figure 7	Prediction results of EBAY by GAN framework	13
Figure 8	Prediction results of EBAY by GRU	13
Figure 9	Prediction results of EBAY by LSTM	13
Figure 10	Prediction results of SBUX by GAN framework	13
Figure 11	Prediction results of SBUX by GRU	14
Figure 12	Prediction results of SBUX by LSTM	14

List of Tables

		Page
Table 1	The details of loss function for the discriminator, the generator and GAN framework	9
Table 2	Comparison of performances of the three models on SBUX	14
Table 3	Comparison of performances of the three models on AAPL	14
Table 4	Comparison of performances of the three models on EBAY	14

1.) Abstract

Deep learning is a subset of machine learning, used by academics and professionals alike, to process unstructured datasets. There have been multiple studies that employed various deep learning mechanisms for the purpose of stock price prediction. Prima facie, it might appear as though a more complex framework like GAN would produce a better prediction result. However, based on previous literature, the efficacy of the GAN framework compared to GRU, and LSTM has been inconclusive within the domain of stock price prediction.

Hence, the objective of this paper is to compare various deep learning models, namely GRU, LSTM, and GAN, and to evaluate the performance of these models when forecasting the prices of various stocks one day after the sample period. For GAN, we used the GRU model as the Generator and the CNN model as the Discriminator. We then measured the performance of these models relative to one another, for 3 stocks (AAPL, EBAY, SBUX), using a defined basket of 15 variables.

2.) Introduction

In the era of artificial intelligence, machine learning, and big data, the use of deep learning for the purpose of stock price prediction has gained traction. We want to apply deep learning to stock price prediction, as stock price prediction is a widely researched financial time series prediction, which involves strong market efficacy and a high level of noise (Fama, 1970). Many related works use classic algorithms such as LSTM and ARIMA to predict stock price. Since there are only a few papers that use the GAN framework for the purpose of stock price prediction, and since previous works that employed the GAN framework for the purpose of stock price prediction also chose different models for the Generator and Discriminator within the GAN framework, we decided to test the efficacy of the GAN framework and compare it to GRU and LSTM.

Since stock price is influenced by an array of factors, we incorporated fundamental data from the financial statements of a company (such as earnings per share divided by closing price and book value per share divided by earning price), economic variables (such as Gold Price and the S&P 500 Index), and technical indicators (such as RSI and MACD). A total of 15 variables were used, over a period of 10 years, from January 2013 to December 2022.

3.) Literature Review

Previous works showed inconsistent results, when comparing the efficacy of the GAN framework, to other deep learning models such as LSTM. Moreover, these papers also used different models for the Generator and Discriminator within the GAN framework. A study that compared the efficacy of the GAN framework (using LSTM as the model for the Generator and CNN as the model or the Discriminator) and the LSTM model when predicting whether stock price would increase one day after the sample period, showed that the accuracy of the GAN framework (72.68%) and the LSTM model (74.16%) were comparable (Ricardo and Carrillo, 2019). However, another similar study showed that the GAN framework (using LSTM as the model for the Generator and MLP as the model for the Discriminator) performs better than the LSTM model when forecasting the one day closing price of stocks. In this study, the accuracy of the GAN framework was 75.54%, while the accuracy of the LSTM model was 68.59% (Kang et al., 2019).

4.) Theoretical Background

4.1) Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is a more advanced version of Recurrent Neural Network (RNN) that solves the vanishing gradients, exploding gradients, and long-term dependency problem in RNN. Unlike a traditional feed-forward neural network, it has been shown that LSTM models can retain and utilize information over longer sequences, as compared to standard RNNs, as it includes feedback connections (Shiri et al., 2023).

The equations for the gates and cells in LSTM are expressed as follows:

Input gate $i_t = \sigma(w_i[h_{t-1}, x_t] + b_i)$

Forget gate $f_t = \sigma\big(w_f[h_{t-1}, x_t] + b_f\big)$

Output gate $o_t = \sigma(w_o[h_{t-1}, x_t] + b_o)$

σ: represents the sigmoid function

w_x: weight for the respective gate(x) neurons

h_{t-1}: output of previous LSTM block (at timestamp t - 1)

x_t: input at current timestamp t

b_x: biases for the respective gates(x)

Cell state (memory) at timestamp t $c_t = f_t * c_{t-1} + i_t * \widetilde{c_t}$

Candidate cell state at timestamp $\mathrm{t}\widetilde{c_t} = \tan \mathrm{h}(w_c[h_{t-1}, x_t] + b_c)$

Final Output $h_t = o_t * \tan \mathrm{h}(c^t)$

Gates in the LSTM model make use of sigmoid activation functions (**Figure 1A in Appendix**), which produces an output of 0, 1 or a value between. An output of 0 means that the gate is blocking everything, while an output of 1 means that the gate allows everything to pass through it.

The basic components of LSTM include an input gate, an output gate, and a forget gate (Lin et al., 2021) as shown in **Figure 2A in Appendix**. The purpose of these gates is to control the flow of information within the LSTM model. The input gate decides on how the internal state will be updated, based on both current input and the previous internal state. The forget gate decides on how much of the previous state should be forgotten. The output gate controls the effect of the internal state on the system (Fang et al., 2021).

Figure 3A in Appendix shows the inputs and the outputs of the LSTM Model at any timestamp t. The input of the LSTM model for this timestep is represented by x_t and the output of the LSTM model for this timestep is represented by y_t. h_(t-1) and c_(t-1) are the inputs taken from the LSTM model of the previous timestep t-1. h_t and c_tare also generated as the inputs for the LSTM model of the next timestep t+1.

Bi-LSTM (Bidirectional Long Short-Term Memory) consists of two LSTM layers and uses bidirectional processing to process sequential data simultaneously in two directions, both forward and backward as shown in **Figure 4A in Appendix**. During forward processing, the input sequence is fed into the forward LSTM layer from the first to the last step (Anish, 2023). During backward processing, the input sequence is also fed into the backward LSTM layer, but in reverse order, from the last to the first step (Anish, 2023). Both the forward and backward LSTM, based on the current input and the previous hidden state and memory cell, will update its memory cell and compute its hidden state. Eventually, the hidden

states of each LSTM layer are combined at each time step, when both forward and backward processes are complete (Anish, 2023).

The benefit of using a Bi-LSTM model is that it is able to better capture dependencies in the input sequence, since it captures both the context that comes before a specific time step and the context that follows (Anish, 2023).

4.2) Gated Recurrent Unit (GRU)

GRU, similar to LSTM, is able to selectively remember and forget information over time. Thus, GRU is also suitable for the modelling of sequential data. However, the most significant difference between GRU and LSTM is the way the model handles the memory cell. In the LSTM model, the memory cell is updated using the input, output, and forget gates. In the GRU model, the memory cell is updated using the candidate hidden state, which in turn is updated using the reset and update gates (**Figure 5A in Appendix**). The reset gate decides how much of the previous hidden state to forget, and the update gate decides how much of the candidate activation vector should be incorporated into the new hidden state (Zhang et. al., 2021). The reset gates capture short-term dependencies in sequences, while the update gates capture long-term dependencies in sequences.

Update gate for timestep t $z_t = \sigma(w_{xz}x_t + w_{hz}h_{t-1} + b_z)$ b_r and b_z are bias parameters

Reset gate for timestep t $r_t = \sigma(w_{xr}x_t + w_{hr}h_{t-1} + b_r)$

For both gates, x_t is multiplied by its own weight w_{xr} or w_{xz} , and h_{t-1} (which holds information related to the previous t-1 units) is multiplied by its own weight w_{hz} or w_{hr}.

Candidate hidden state for timestep t $\widetilde{H_t} = \tanh(x_t w_{xh} + (R_t \odot H_{t-1})w_{hh} + b_h)$

w_{xh} and w_{hh} are weight parameters, b_h is the bias, and $\odot$ is the Hadamard (elementwise) product operator. In the candidate hidden state, the tanh activation function is applied.

The purpose of the candidate hidden state (**Figure 6A in Appendix**) is to combine information from the input and the previous hidden state. The GRU computes a candidate hidden state at each time step, which is then used to update the hidden state for the following time step (Zhang et. al., 2021)

Hidden step for timestamp t $H_t = z_t \odot H_{t-1} + (1 - Z_t) \odot \widetilde{H_t}$

In the hidden step, the update gate $z_t$determines how much of the new hidden state H_t matches the previous hidden state H_{t-1} (**Figure 7A in Appendix**), and the new candidate state $\widetilde{H_t}$ (Zhang et. al., 2021). When the value of the output gate z_t is 1, we retain the hidden step of the previous timestamp H_{t-1}, and we ignore the input x_t. This would be equivalent to excluding information related to timestep t. On the other hand, when the value of the output gate $z_t$is 0, the hidden state of timestamp t would approach the candidate hidden state $\widetilde{H_t}$(Zhang et. al., 2021).

4.3) Convolutional neural network (CNN)

CNN is a deep learning model that is used to process data, such as images, that have grid patterns. It consists of 3 layers: convolution, pooling, and fully connected layers as shown in **Figure 8A in Appendix**. The purpose of the first two layers, convolution and pooling (eg. max pooling), is to perform feature extraction. A convolution layer is a key component of a CNN model as it is able to apply a small grid of parameters referred to as a kernel, to an array of numbers. The third fully connected layer then maps the extracted features to the final output (Yamashita, 2018). The CNN model's performance is

calculated with a loss function on a training dataset, through forward propagation, using learnable parameters such as kernels and weights. These learnable parameters are then updated through backward propagation with the gradient descent optimization algorithm, based on the loss value (Yamashita, 2018)

4.4) Generative Adversarial Network (GAN)

Generative Adversarial Network (GAN) is a framework for estimating generative models via an adversarial process developed by Goodfellow et al. (2014). It consists of two neural networks that compete against each other: a Generator and a Discriminator. The GAN framework aims to train the Generative model. The Generator tries to generate fake data that looks like real data and feeds the fake data with real data to the Discriminator. The Discriminator then tries to discriminate between real data and fake data. The probability of the Discriminator being able to identify real or fake data correctly is used to train both the Generator and the Discriminator. Both neural networks are trained until the Discriminator is unable to differentiate between the real and the fake data. The Generator will then be at its optimal data generating point.

The Discriminator returns a numeric value close to 1 if it classifies the data as real and close to 0 if it classifies the data as fake. Log-likelihood is used to measure the accuracy of the Discriminator. $log(D(x))$ represents the log-likelihood that the Discriminator accurately categorizes real data and $log(1 - D(G(z_i)))$ is the log-likelihood that the Discriminator correctly determines generated data as fake.

With the definition of $\log(D(x))$ and $\log(1 - D(G(z_i)))$, GAN can be modeled as a two-player minimax game whereby each player reduces their losses or increases the costs of the other players. Minimax refers to minimizing the loss in the Generator and maximizing the loss in the Discriminator. Consider a range of real data ($x \sim p_{data}$(x)) and a range of generated data ($z \sim p_z(z)$), the equation describing minimax game is given as:

$$\min_G \max_D (G, D) = E_{x \sim p_{data}(x)}\left[log\left(D(x)\right)\right] + E_{z \sim p_z(z)}[log(1 - D(G(Z_i)))]$$

Where $\boldsymbol{\min_G}$ represents the parameters of the Generator that are adjusted to minimize $\log(1 - D(G(z_i)))$ and $\boldsymbol{\max_D}$ represents the parameters of the Discriminator that are optimized to maximize both $\log(D(x))$ and $\log(1 - D(G(Z_i)))$. Therefore, the loss function for optimizing the Generator (J_G) and the Discriminator (J_D) can be given as:

$$J_G = G - loss = \min_G E_{z \sim p_z(z)}\left[log\left(1 - D\left(G(Z_i)\right)\right)\right] = \min_G -E_{z \sim p_z(z)}\left[log\left(D\left(G(Z_i)\right)\right)\right]$$

$$J_D = D - loss = \max_D E_{x \sim p_{daya}(x)}[\log(D(x)] + \max_D E_{z \sim p_z(z)}[\log\left(1 - D\left(G(z)\right)\right]$$

To change the maximization process to a minimization process, the equation for J_D can be rewritten as:

$$J_D = D - loss = \min_D -E_{x \sim p_{daya}(x)}[\log(D(x)] - \min_D E_{z \sim p_z(z)}[\log\left(1 - D\left(G(z)\right)\right]$$

Figure 10A of Appendix visualizes the overall training process in the GAN framework. The black, dotted line represents the distribution of real data, the green line shows the distribution of fake data, and the blue dashed line represents discriminative distribution. Step (a) shows an adversarial pair that is near convergence. The distribution of fake data is similar to the distribution of real data and the

Discriminator is a partially accurate classifier. In step (b), the Generator is fixed, and the Discriminator is trained to discriminate fake data from real data. After the Discriminator is updated, the gradient from the Discriminator is used to guide the Generator to generate data is that more likely to be classify as real in step (c). After several steps of training, the models will reach a point at which both cannot improve because the distribution of the fake data is the same as the distribution of the real data as shown in step (d). The Discriminator cannot distinguish between real and fake data and becomes a flat line.

5.) Dataset and Features

5.1) Datasets

In this project, the main objective is to compare the predictive performance for stock price prediction of three different deep-learning models namely, Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), and Generative Adversarial Network (GAN). Three stocks, including Apple Inc. (AAPL), eBay Inc. (EBAY), and Starbucks Corporation (SBUX), were chosen from different industries in US stock markets with the objective of observing the influence of these different industries on experimental results. The targeted variable of the model is closing price. The historical closing price of stocks in the period January 1, 2013 to December 31, 2022 were collected from the yfinance library.

5.2) Features

In stock price prediction, selecting features is a vital process that significantly affects the accuracy of the model. Using variables that have predictive power on stock price results in higher predictive performance, while including irrelevant inputs may lead to an increase in the complexity of the model, an overfitting problem, and an increase in computational cost.

Htun et al. (2023) summarized that there are three structure-type variables that are commonly used as features in stock price prediction, namely basic features, technical indicators, and fundamental indicators. Basic features include the historical open, high, low, close, and volume data of the stock, while technical indicators are the information that is subtracted from the historical price series by mathematical formulas. Fundamental indicators range from macroeconomic factors, such as interest rates, to microeconomic factors, such as the information of the company.

There have been less studies so far that applied more than one type of feature, as compared to studies that applied only one type of feature. In addition, there are three studies that applied multiple feature types and obtained accurate predictions. These two reasons were the motivation behind why we selected 15 representative variables from all three categories and used them as inputs for our chosen deep learning models. A summary of the features we used is shown in **Table 1 in Appendix** and the variables that we selected for each feature type are as follows:

5.2.1) Basic features

Rana et al. (2019) applied feature selection algorithms on basic features with the objective of measuring feature importance of opening price, closing price, low price, high price and volume for a closing price prediction model. The result showed that closing price is the most significant feature with the feature importance value of around 0.45 while for other variables, the value ranged from around 0.12 to 0.15. Therefore, we decided to use only closing price as a feature for the deep learning models that we employed in this project.

5.2.2) Technical indicators

Many studies have shown that technical indicators have predictive power when being used as inputs of machine learning models for stock price prediction. For instance, Qolipour et al. (2021) compared the predictive performance of machine learning algorithms (namely decision trees, gradient boosted tree and random forest) when using only basic features as inputs and when using both basic features and technical indicators as inputs. The results showed that when the technical indicators are included, all of the models provided a higher value of accuracy.

In this project, we selected and calculated 4 commonly used technical indicators from closing price and used them as features to capture information regarding stock price movement. The technical indicators we selected are the 21-day simple moving average to capture trend of price movement, MACD (12-day short EMA and 26-day long EMA) to extract the changes in short-term trends compared to long-term trends, RSI (14-day window length) to account for strength of trend and Bollinger band (upper band and lower band) to measure volatility.

5.2.3) Fundamental indicators

Fundamental indicators can be divided into 2 groups: microeconomic factors and macroeconomic factors. A previous study showed that using both microeconomic factors such as financial ratios and technical indicators as inputs of the model significantly improved model performance as compared to using technical indicators as inputs alone (Namdari et al., 2018). In this project, we selected 4 financial metrics from quarterly financial statements that demonstrate a company's profit-making capability and its overall financial health. The metrics comprise earning per share (TTM), net profit margin (TTM), book value and debt-to-equity ratio. As these variables are updated quarterly, we divided them by closing price to make them move on a daily basis. The report date rather than the end-of-quarter date was utilized as the reference date, to reflect the date when the public becomes aware of the updated fundamental data, ensuring that our analysis incorporates the most current and publicly available information. All financial statement data were obtained from Wharton Research Data Service (WRDS).

Macroeconomic factors are also proven to have the ability to predict stock price movements. Several studies have applied various macroeconomic factors such as gold price, oil price, and indices related to monetary policy as features for machine learning models. For instance, Kohli et al. (2019) used basic features and macroeconomic factors, including commodity price and foreign exchange, as explanatory variables and yielded a model that returned an accuracy of more than 70%. For our analysis, we used 5 macroeconomic factors as explanatory variables. The factors included Federal funds rate, Oil closing price, Gold closing price, S&P500 and Nasdaq100. All data was collected from the yfinance library.

6.) Methodology

6.1.) Data Preprocessing

The reshaping of the raw time series data is to prepare the data for meaningful forecasting. The key aspects of this data preprocessing phase are as shown in **Figure 1**:

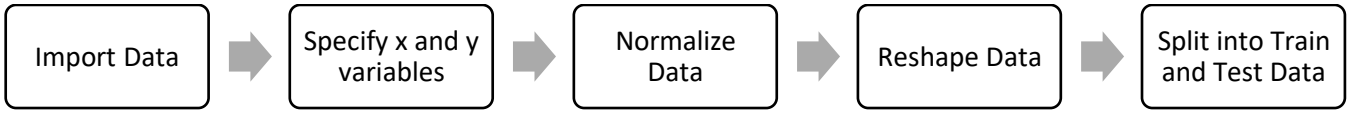

Figure 1: Data preprocessing procedures

6.1.1.) Import Data

The dataset includes the historical closing price of three stocks, Starbucks, Apple, and eBay over 10 years, from 2013 to 2022. It also consists of 14 features which are shown in **Table 1A in Appendix**. The formulas for calculating technical indicators are shown in **Table 2A in Appendix**.

6.1.2.) Define X and Y variables

The x-variable consists of the closing price of the stock and 14 features while the y-variable consists of the closing price only.

6.1.3.) Normalization

Normalization was then applied to both the 16 input features and the targeted variable. This crucial step ensures that all input features are on a consistent scale, promoting faster convergence, mitigating issues related to vanishing or exploding gradients, and enhancing the model's generalization to unseen data. The MinMax scaler, configured with a range spanning from -1 to 1, was chosen for normalizing data in this project because of its capability to effectively manage outlier data, ensuring a robust scaling transformation. The formula for MinMax scaler is as follows:

$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}} \qquad and \qquad Y_{scaled} = \frac{Y - Y_{min}}{Y_{max} - Y_{min}}$$

6.1.4.) Reshaping Data and Train test split

In this project, a predictive model was developed using a sequence of three-day historical data (T-1, T-2, T-3) to forecast the stock price on the subsequent day (T). To facilitate this, the dataset underwent a reshaping process, wherein the 3-day features' data was formatted as input, and the stock price of the following day served as the output for model training and testing.

The dataset then underwent a division into training and testing sets, where 70% of the dataset was allocated for training of the models and the remaining 30% was reserved for testing of the models. This partition resulted in the training set encompassing historical price data spanning from the start of 2013 to the end of 2019, comprising a total of 1752 observations, while the testing set encompassed data from the beginning of 2020 to the end of 2022, comprising a total of 751 observations.

6.2) Generator and Discriminator for GAN framework

In the GAN framework, we designated the Gated Recurrent Unit (GRU) as the generator and the Convolutional Neural Network (CNN) as the discriminator. The generator processes 3D input data comprising features, batch size and input-step, and produces output with batch size and output-step dimensions. To optimize the generator's performance, we applied three layers of GRU, where the number of neurons are arranged in a descending order, 1024, 512 and 256. Subsequently, two Dense layers were incorporated.

The discriminator component within our GAN framework utilizes CNN with the primary objective of distinguishing between genuine and fabricated input data. The input data would either be data generated from the generator or from the original dataset. The discriminator consists of three distinct 1D Convolution layers, each containing 32, 64, and 128 neurons, respectively. **Figure 11A in Appendix** visualizes the components of the generator and discriminator used in GAN framework in our analysis.

Additionally, there are three Dense layers incorporated at the end, with neuron counts of 220, 220, and 1. Throughout these layers, the activation function employed is the Leaky Rectified Linear Unit (ReLU),

except for the output layer. In this case, for GAN, the output layer uses the Sigmoid activation function to yield a scalar output of either 0 (indicating fake data) or 1 (indicating real data). (Lin H et al. ,2021)

The discriminator assumes a pivotal role in distinguishing between genuine and fabricated data. When the generator produces synthetic data, the discriminator's task is to identify which datapoints are "fake". This involves calculating a loss that quantifies how effectively the discriminator distinguishes between generated and real data. Simultaneously, the discriminator receives real, historical stock price data to classify it as "real". The discriminator's learning is shaped by its ability to accurately classify both types of data. The generator, in turn, refines its parameters based on the feedback received from the discriminator, aiming to produce synthetic data that becomes progressively harder for discriminator to classify as fake. Through this iterative and adversarial training process, the GAN converges towards a state where the generator generates synthetic data that closely mirrors the characteristics of real data, contributing to the overall success of this framework.

6.3) GAN Framework Structure

Incorporating the architectural designs of both our generator and discriminator, we have seamlessly merged these components to form our envisioned GAN framework. Within our GAN framework, we employed the binary cross-entropy method to compute losses for both the generator and discriminator. (Lin H et al. ,2021)

Cross-entropy is a mathematical idea that finds extensive applications in a range of domains, such as information theory, machine learning, and statistics. It is a particularly common choice as a loss function in machine learning models, especially when dealing with classification tasks.

At its core, cross-entropy serves as a metric to gauge the dissimilarity between the predicted probability distribution and the actual probability distribution of outcomes.

There are 2 types of cross-entropy, binary cross-entropy, and categorical cross-entropy. In our model, we have used binary cross-entropy (log loss) since we were dealing with 2 possible outcomes. (e.g. 0 or 1). The binary cross-entropy loss equation is written as

$$\boldsymbol{L}(\boldsymbol{y},\hat{y}) = -[\boldsymbol{y}.\boldsymbol{log}\hat{y} + (\boldsymbol{1}-\boldsymbol{y}).\boldsymbol{log}(\boldsymbol{1}-\hat{y})]$$

The discriminator loss equation is achieved by combining both the losses for a real input and fake input. The generator loss is calculated based on the discriminator loss. While training the generator, the discriminator remains unchanged, meaning it can only evaluate the fake input. Consequently, the initial component of the discriminator loss equation becomes zero. The generator's objective is to deceive the discriminator by making it categorize the counterfeit data as genuine. In essence, the generator aims to reduce the impact of the second element in the discriminator loss equation (Lakshmi Ajay, 2022). **Table 1** shows the details of loss equation of the discriminator, the generator and GAN framework.

Model / Framework	Equation
1.) Discriminator	For real input, y=1, substituted into binary cross entropy loss equation $\boldsymbol{L}(\boldsymbol{1},\hat{y}) = -\big[\boldsymbol{1}.\boldsymbol{log}\big(\boldsymbol{D}(\boldsymbol{x})\big) + (\boldsymbol{1}-\boldsymbol{1}).\boldsymbol{log}\big(\boldsymbol{1}-\boldsymbol{D}(\boldsymbol{x})\big)\big] = -\boldsymbol{log}\big(\boldsymbol{D}(\boldsymbol{x})\big)$ Where $\hat{y} = \mathrm{D}(\mathrm{x})$, output of discriminator for a real output For Fake output, y=0, substituted into binary cross entropy loss equation

Model / Framework	Equation
	$L(0,\hat{y}) = -\left[0.log\left(D(G(z))\right) + (1-0).log\left(1-D(G(z))\right)\right] = -log(1-D(G(z)))$ Where $\hat{y} = D(G(z))$, output of discriminator with generated data as input. Z = input noise vector provided to the generator. And G(z) = output of generator Combining both the loss equations to get Discriminator Loss Equation $L(Discriminator) = -[log(D(x)) + log(1-D(G(z)))]$ Goal of discriminator is to minimize loss $L(Discriminator) = \min\{-[log(D(x)) + log(1-D(G(z)))]\}$ We transform the equation into a max. equation by removing the "-" sign $L(Discriminator) = \max\{[log(D(x)) + log(1-D(G(z)))]\}$
2.) Generator	$L(Discriminator) = \min\{[log(1-D(G(z)))]\}$
3.) GAN	Combining the Discriminator and Generator Loss equations, The GAN loss equation for a single datapoint can be obtained. $L(GAN) = GminDmax\{[log(D(x)) + log(1-(D(G(z)))]\}$ From the above equation, it can be deduced that the generator minimizes the loss while the discriminator maximizes the loss. GAN Loss Equation for expected values (E) for multiple datapoints is $L(GAN) = GminDmax\{[E_{x(real)}\ log(D(x)) + E_{z(fake)}\ log(1-(D(G(x)))]\}$

Table 1: The details of loss function for the discriminator, the generator and GAN framework

Additionally, in the discriminator, we amalgamated the generated stock price with the historical stock price data from the input steps as shown in **Figure 2** . This inclusion extends the dataset's length and augments the accuracy of the discriminator's classification learning process.

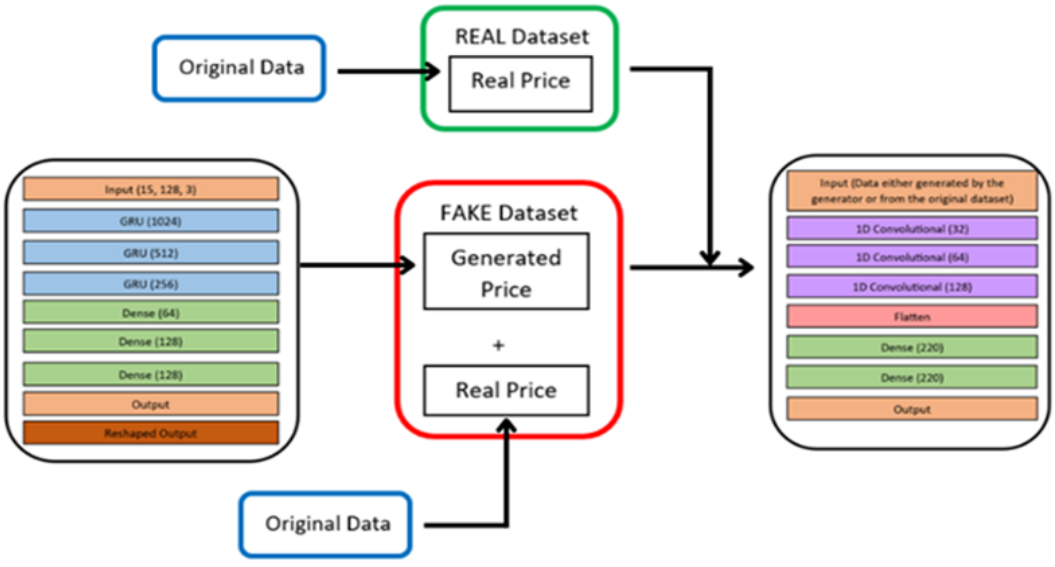

Figure 2: Architecture of GAN framework

6.4) LSTM Model

A Bidirectional LSTM layer with 128 units of neurons is employed, enabling the model to capture patterns in both forward and backward directions. This is particularly beneficial for sequence modeling tasks. As shown in **Figure 12A in Appendix**, two dense layers are utilized, with the first comprising 64 units, and the final layer having units equal to the output dimension which is set as 1. The activation function for these dense layers is linear, as it is the default activation for the dense layer in Keras.

The inclusion of two dense layers allows the model to perform a hierarchical mapping of features learned by the Bidirectional LSTM layer. The first dense layer acts as an intermediary representation, and the second dense layer maps this representation to the output space. This can enable the model to learn both local and global patterns in the sequential data.

The model is trained using the Adam optimizer with a learning rate of 0.001 and Mean Squared Error (MSE) as the loss function. The training process is executed for 50 epochs with a batch size of 64.

6.4. GRU Model

Two GRU layers are employed in the model. As visualized in **Figure 13A in Appendix**, the first GRU layer comprises of 128 units and is set to return sequences, producing an output for each input sequence. The second GRU layer has 64 units, and it follows the first layer without returning sequences to enable the model to distill and consolidate the learned hierarchical features captured by the first layer. The second layer serves as a higher-level representation, focusing on the global context of the input sequence. This hierarchical feature extraction allows the model to learn both local and global patterns in the sequential data with the goal of improving its overall predictive performance.

Similar to the LSTM model, the incorporation of two dense layers enables a hierarchical mapping of learned features from the GRU layers. The first dense layer comprises of 32 units while the second layer is set to be equal as the output dimension which is 1.

The model is compiled using the Adam optimizer with a learning rate of 0.0001 and Mean Squared Error (MSE) as the loss function. The use of the Adam optimizer provides an adaptive learning rate mechanism that contributes to stable and efficient training of the model. The training process is executed for 50 epochs with a batch size of 64.

7.) Experimental Results

The primary objective of this research is to forecast the closing price of stocks for the next day based on a dataset comprising information from the preceding 3 days. In order to enhance the predictive capabilities of our model, we conducted comprehensive feature engineering and identified 15 key features deemed crucial for accurate predictions. The dataset was divided into a training set (70%) and a testing set (30%) to facilitate the assessment of model performance.

7.1) Performance measurements

To evaluate the effectiveness of each model, we employed various performance metrics, each providing unique insights into the model's predictive accuracy:

- **Root Mean Squared Error (RMSE)**
  RMSE measures the square root of the average squared differences between predicted $(\hat{Y}_i)$ and actual (Y_i) values:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} \left(\hat{Y}_i - Y_i\right)^2}{n}}$$

It provides a sense of the typical size of the errors in the predictions. The lower the RMSE, the better the model's performance.

- **Mean Absolute Error (MAE)**
  MAE calculates the average absolute differences between predicted $\left(\hat{Y}_i\right)$ and actual (Y_i) values:

$$MAE = \frac{\sum_{i=1}^{n} \left|\hat{Y}_i - Y_i\right|}{n}$$

  It offers a straightforward measure of prediction accuracy, with lower MAE values indicating better model performance.

- **Mean Absolute Percentage Error (MAPE)**
  MAPE expresses the average percentage difference between predicted $\left(\hat{Y}_i\right)$ and actual (Y_i) values:

$$MAPE = \frac{1}{n}\sum_{i=1}^{n} \left|\frac{\hat{Y}_i - Y_i}{Y_i}\right| \times 100$$

  It is particularly useful for interpreting errors in terms of the relative size of the predicted values. Lower MAPE values signify more accurate predictions.

- **Mean Squared Logarithmic Error (MSLE)**
  MSLE measures the average of the logarithmic differences between predicted $\left(\log\left(\hat{Y}_i + 1\right)\right)$ and actual $\left(\log\left(Y_i + 1\right)\right)$ values:

$$MSLE = \frac{\sum_{i=1}^{n} \left(\log\left(\hat{Y}_i + 1\right) - \log\left(Y_i + 1\right)\right)^2}{n}$$

  It is especially useful when the target variable has exponential growth patterns. Lower MSLE values indicate better accuracy in capturing the magnitude of differences on a logarithmic scale.

- **R-squared (R2)**
  R-squared assesses the proportion of the variance in the dependent variable (closing stock prices) that is predictable from the independent variables (features):

$$R^2 = 1 - \frac{\sum_{i=1}^{n} \left(Y_i - \hat{Y}_i\right)^2}{\sum_{i=1}^{n} \left(Y_i - \bar{Y}\right)^2}$$

  where $\bar{Y}$ is the mean of the actual values. A higher R-squared value indicates a better fit of the model to the data, with 1.0 representing a perfect fit.

7.2) Experimental results

In the Generative Adversarial Network (GAN) approach, the model was trained with a learning rate of 0.0016, 165 epochs, and a batch size of 128. For the Long Short-Term Memory (LSTM) model, a learning rate of 0.001, 50 epochs, and batch size of 64 were employed. Similarly, the Gated Recurrent Unit (GRU) model utilized a learning rate of 0.0001, 50 epochs, and a batch size of 64.

The hyperparameter tuning process involved several iterations of trial and error, ultimately leading to the determination of the most optimal values for achieving the highest predictive accuracy in our models.

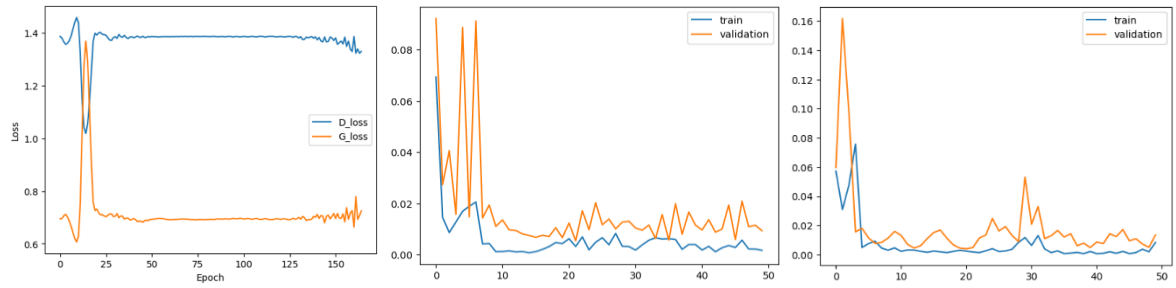

Figure 3: Observable patterns in the loss functions of GAN (left), LSTM (middle) and GRU (right)

In contrast to the observable patterns in the loss functions of Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), where the loss functions of both are decreasing and become stabilized, the loss function of Generative Adversarial Networks (GANs) exhibit a distinct and somewhat counterintuitive result. Unlike the gradual convergence observed in the loss functions of LSTM and GRU, the Discriminator (D_loss) in GANs appears to plateau at a relatively high point. This phenomenon can be attributed to the inherent adversarial nature of GANs, where the generator and discriminator engage in a constant competitive interplay. Improvements in one component result in higher losses for the other, creating a dynamic struggle. However, over the course of training, both the discriminator and generator losses tend to converge to stable, seemingly permanent values. This convergence signifies an equilibrium, indicating that the generator has learned to produce realistic samples, rendering the discriminator unable to effectively distinguish between real and generated data. To successfully train all of the above models, we had to adjust the value of hyperparameters, pick the right network architecture, and continuously monitor their respective loss functions.

Figure 4: Prediction results of AAPL by GAN framework

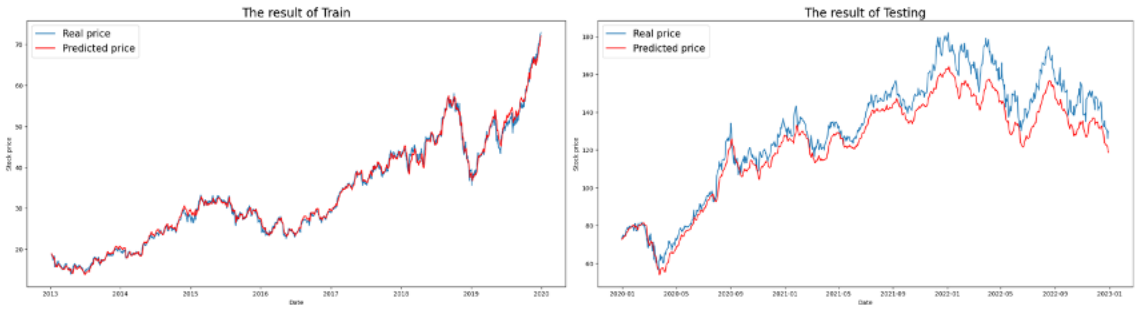

Figure 5: Prediction results of AAPL by GRU

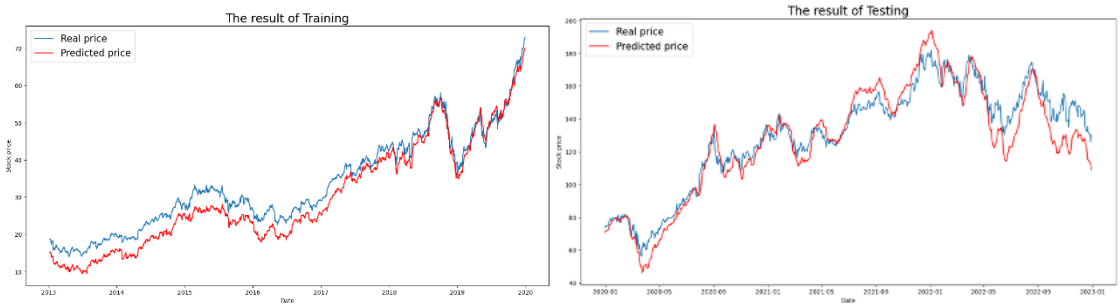

Figure 6: Prediction results of AAPL by LSTM

Figure 7: Prediction results of EBAY by GAN framework

Figure 8: Prediction results of EBAY by GRU

Figure 9 Prediction results of EBAY by LSTM

Figure 10: Prediction results of SBUX by GAN framework

Figure 11: : Prediction results of SBUX by GRU

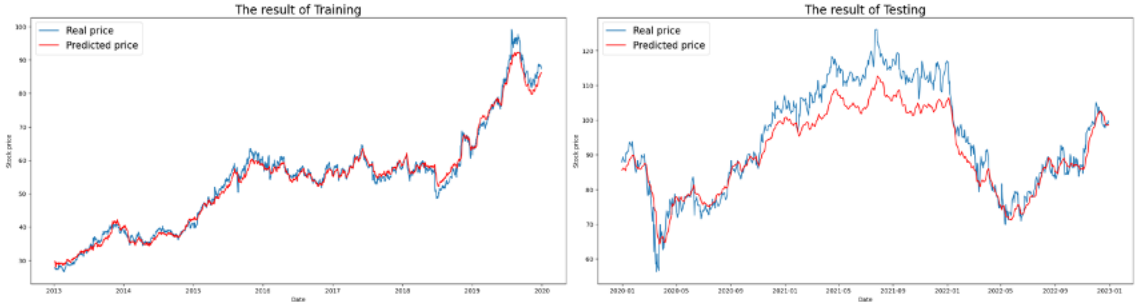

Figure 12: Prediction results of SBUX by LSTM

Performance Comparisons

SBUX	GAN		LSTM		GRU	
	training	testing	training	testing	training	testing
RMSE	1.27	3.01	1.7	5.758	1.83	4.825
MAE	0.95	2.2	1.324	4.557	1.59	3.866
MAPE	0.0194	0.0249	0.0249	0.0458	0.033	0.0455
MSLE	0.00063	0.001	0.000916	0.00436	0.001577	0.00325
R^2	0.993	0.962	0.987	0.86	0.985	0.902

Table 2: Comparison of performances of the three models on SBUX

AAPL	GAN		LSTM		GRU	
	training	testing	Training	testing	training	testing
RMSE	1.079	8.3975	3.811	9.401	0.8843	9.96
MAE	0.847	6.796	3.484	7.309	0.6446	8.314
MAPE	0.0286	0.05	0.1348	0.0574	0.02	0.06
MSLE	0.00135	0.00351	0.02992	0.00608	0.00064	0.005
R^2	0.992	0.925	0.9096	0.906	0.9951	0.895

Table 3: Comparison of performances of the three models on AAPL

EBAY	GAN		LSTM		GRU	
	training	testing	Training	testing	training	testing
RMSE	1.079	3.198	2.575	6.095	1.491	3.3557
MAE	0.873	2.63	2.1072	5.25	1.2518	2.7523
MAPE	0.0327	0.0518	0.0844	0.1022	0.0493	0.0546

EBAY	GAN		LSTM		GRU	
	training	testing	training	testing	training	testing
MSLE	0.00152	0.00364	0.0126	0.0125	0.00325	0.0042
R^2	0.973	0.927	0.846	0.734	0.948	0.9192

Table 4: Comparison of performances of the three models on EBAY

The table presented above provides a comprehensive comparison of training and testing results for the various models across the 3 stocks we chose. It can be seen that Generative Adversarial Network (GAN) consistently outperforms other models in terms of testing accuracy. However, Gated Recurrent Unit (GRU) demonstrates better performance in certain training scenarios compared to GAN and LSTM.

For Long Short-Term Memory (LSTM) model, it was observed that a smaller batch size provides better results. Therefore, a batch size of 64 was selected, and the Adam optimizer with a learning rate of 0.001 was employed. Nevertheless, the LSTM model lagged behind GRU and GAN in terms of accuracy.

In the case of GRU, the training results are comparable to our proposed GAN framework. Nevertheless, GAN exhibits higher accuracy during testing, indicating its robustness across different datasets. This is because after carrying out hyperparameter tuning, we identified consistent accuracy patterns across the three stocks we chose, when we analyzed them using the GAN framework.

In summary, these findings indicate that the GAN network architecture excels in capturing time series representations and has a superior accuracy as compared to traditional LSTM and GRU methods. These results underscore the potential of GANs to enhance predictive modeling for financial data analysis. It is important to note, however, that the GAN framework's complex architecture necessitates extensive trial and error during hyperparameter tuning. Future studies can explore more effective approaches to streamline this process for the purpose of optimizing the performance of the GAN framework.

References

Malkiel, B. G., & Fama, E. F. (1970). Efficient capital markets: a review of theory and empirical work. *Journal of Finance*, 25(2), 383–417.

Shiri, F., Perumal, T., Mustapha, N., Mohamed, R., Ahmadon, M. A. B., & Yamaguchi, S. (2023). A Survey on Multi-Resilient Activity recognition in Smart Environments.

Ricardo, A., & Carrillo, R. (2019). Generative Adversarial Network for Stock Market price Prediction. *Stanford University*.

Zhang, K., Zhong, G., Dong, J., Wang, S., & Wang, Y. (2019). Stock market prediction based on generative adversarial network. *Procedia Computer Science*, 147, 400-406.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. C., & Bengio, Y. (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13, 2014, Montreal, Quebec, Canada* (pp. 2672–2680).

Lin, H., Chen, C., Huang, G., & Jafari, A. (2021). Stock price prediction using Generative Adversarial Networks. *Journal of Computer Science*, 17(3), 188-196. https://doi.org/10.3844/jcssp.2021.188.196

Fang, W., Chen, Y., & Xue, Q. (2021). Survey on research of RNN-based, spatio-temporal sequence prediction algorithms. *Journal on Big Data*, 3(3), 97.

Ihianle, I., Nwajana, A., Ebenuwa, S., Otuka, R., Owa, K., & Orisatoki, M. (2020). A Deep Learning Approach for Human Activities Recognition From Multimodal Sensing Devices. *IEEE Access*, 8, 179028-179038. https://doi.org/10.1109/ACCESS.2020.3027970

Yamashita, R., Nishio, M., Do, R. K. G., et al. (2018). Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9, 611–629. https://doi.org/10.1007/s13244-018-0639-9

Abedi, M., Hempel, L., Sadeghi, S., Kirsten, T. (2022). GAN-Based Approaches for Generating Structured Data in the Medical Domain. *Applied Sciences*, 12, 7075. https://doi.org/10.3390/app12147075

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative Adversarial Networks. *Advances in Neural Information Processing Systems*, 3. https://doi.org/10.1145/342262

Sefidian. (2022, August 18). *A guide on regression error metrics (MSE, RMSE, MAE, MAPE, sMAPE, MPE) with Python code*. Sefidian. Retrieved from https://sefidian.com/2022/08/18/a-guide-on-regression-error-metrics-with-python-code/

Ajay, L. (2022). Decoding the Basic Math in GAN — Simplified Version. *Towards Data Science*. Retrieved from https://towardsdatascience.com/decoding-the-basic-math-in-gan-simplified-version-6fb6b079793

Htun, H. H., Biehl, M., & Petkov, N. (2023). Survey of Feature Selection and Extraction Techniques for Stock Market Prediction. *Financial Innovation*, 9, Article 26. https://doi.org/10.1186/s40854-022-00441-7

Rana, M., Uddin, M. M., & Hoque, M. M. (2019). Effects of activation functions and optimizers on stock price prediction using LSTM recurrent networks. In *CSAI, Beijing, China* (pp. 354–358).

Qolipour, F., Ghasemzadeh, M., & Mohammad-Karimi, N. (2021). The predictability of tree-based machine learning algorithms in the big data context. *International Journal of Engineering*, 34(01), 82–89.

Namdari, A., & Li, Z. S. (2018). Integrating Fundamental and Technical Analysis of Stock Market through Multi-layer Perceptron. In *2018 IEEE Technology and Engineering Management Conference (TEMSCON)* (pp. 1-6). Evanston, IL, USA. https://doi.org/10.1109/TEMSCON.2018.8488440.

Barth, J., Weng, B., Martinez, W., Tsai, Y. T., Li, C., Lu, L., & Megahed, F. (2018). Macroeconomic Indicators Alone can Predict the Monthly Closing Price of Major U.S. Indices: Insights from Artificial Intelligence, Time-Series Analysis and Hybrid Models. *Applied Soft Computing*, 71. https://doi.org/10.1016/j.asoc.2018.07.024.

Kohli, P. P. S., Zargar, S., Arora, S., & Gupta, P. (2019). Stock prediction using machine learning algorithms. In *Applications of Artificial Intelligence Techniques in Engineering* (Advances in Intelligent Systems and Computing 698) (pp. 405–414).

Anish, N. (2023). Understanding Bidirectional LSTM for Sequential Data Processing. *Medium*. Retrieved from https://medium.com/@anishnama20/understanding-bidirectional-lstm-for-sequential-data-processing-b83d6283befc

Zhang, A., Lipton, Z., Li, M., & Smola, A. (2021). Dive into Deep Learning.

Salehi, P., Chalechale, A., & Taghizadeh, M. (2020). Generative adversarial networks (GANs): An overview of theoretical model, evaluation metrics, and recent developments. *arXiv preprint arXiv:2005.13178*.

Appendix

A.) Figures

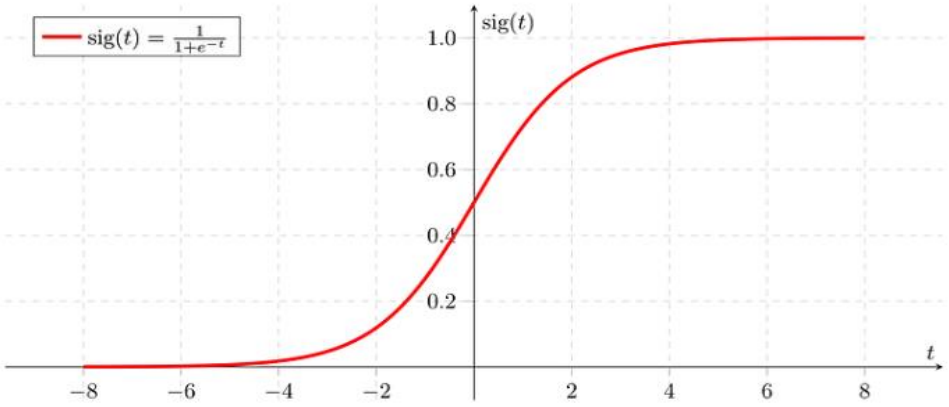

Figure 1A: Graph of Sigmoid Function

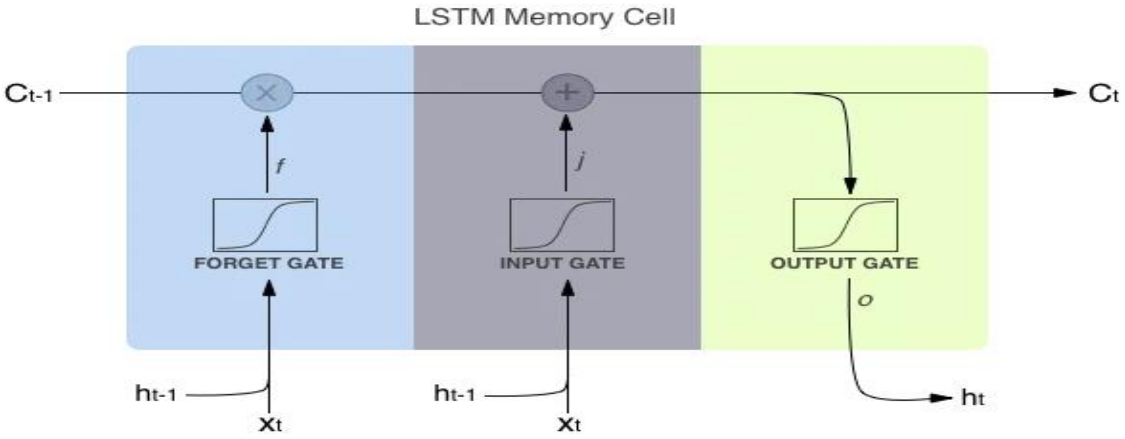

Figure 2A: LSTM Memory Cell

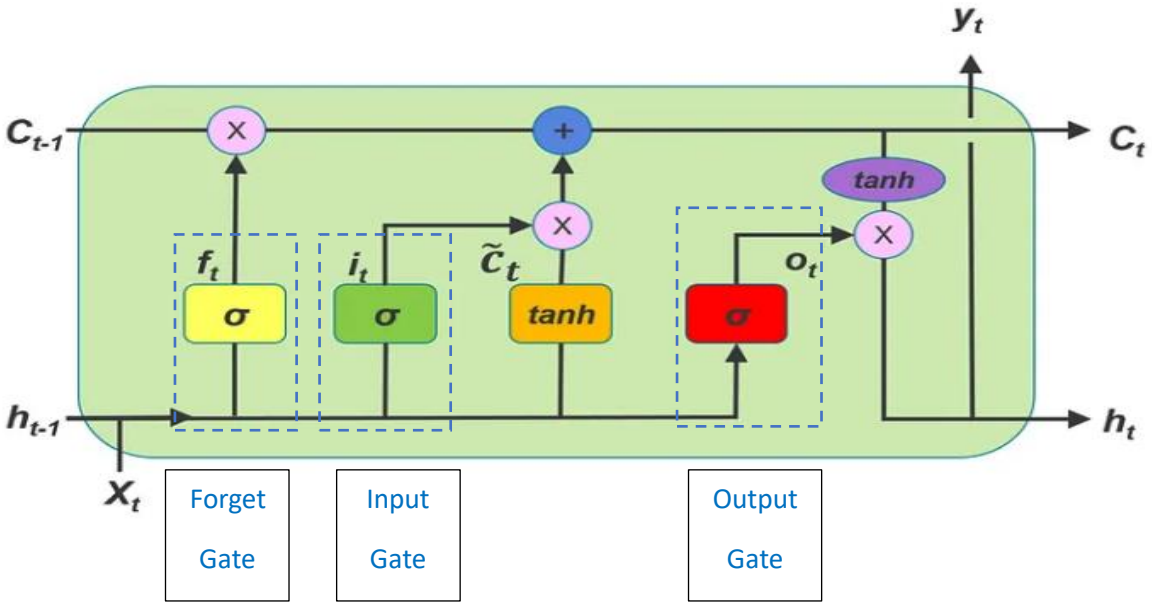

Figure 3A: LSTM Model at any Timestamp t

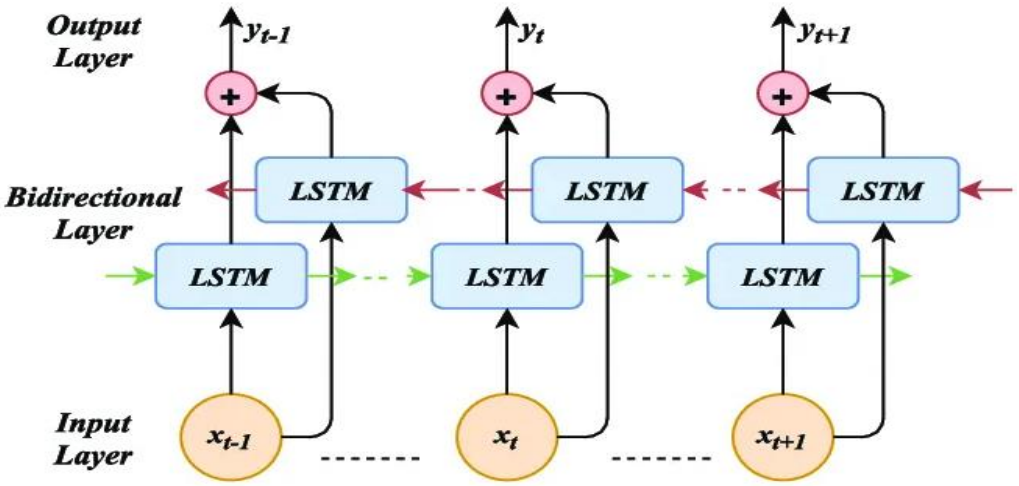

Figure 4A: Bi-LSTM (Ihianle, 2020)

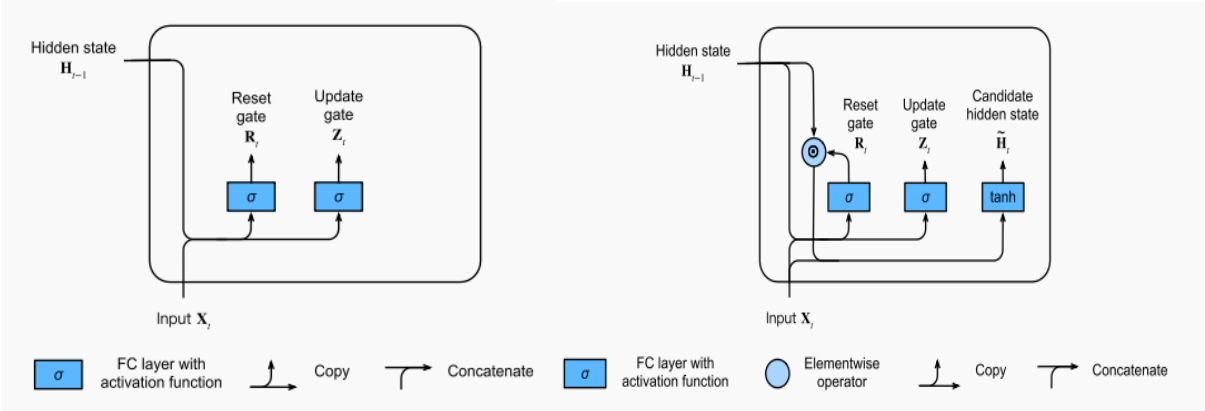

Figure 5A: Update and Reset Gate of the GRU Model (Zhang et. al., 2021)

Figure 6A: Candidate Hidden State of the GRU Model

(Zhang et. al., 2021)

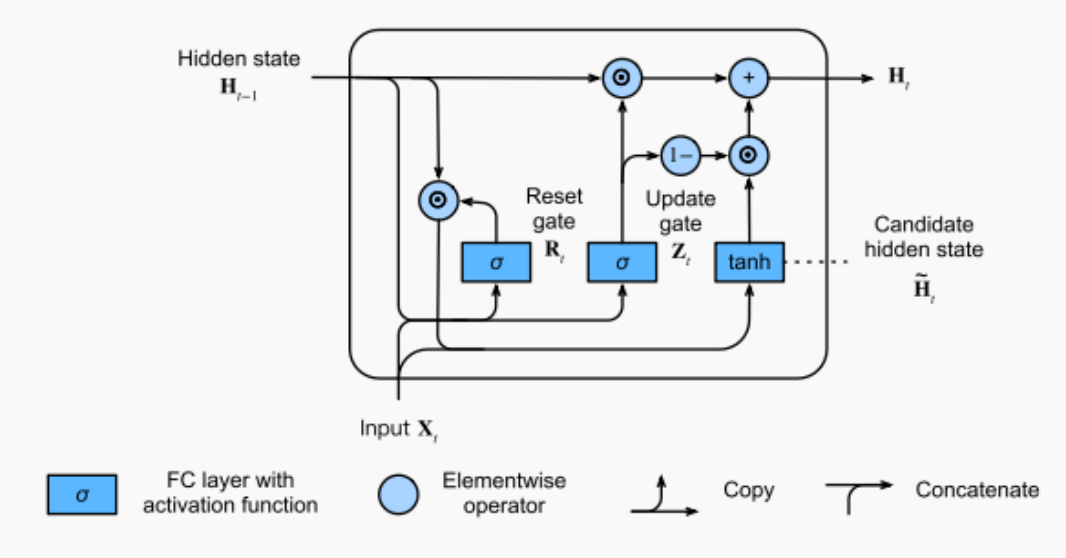

Figure 7A: Hidden State of the GRU Model

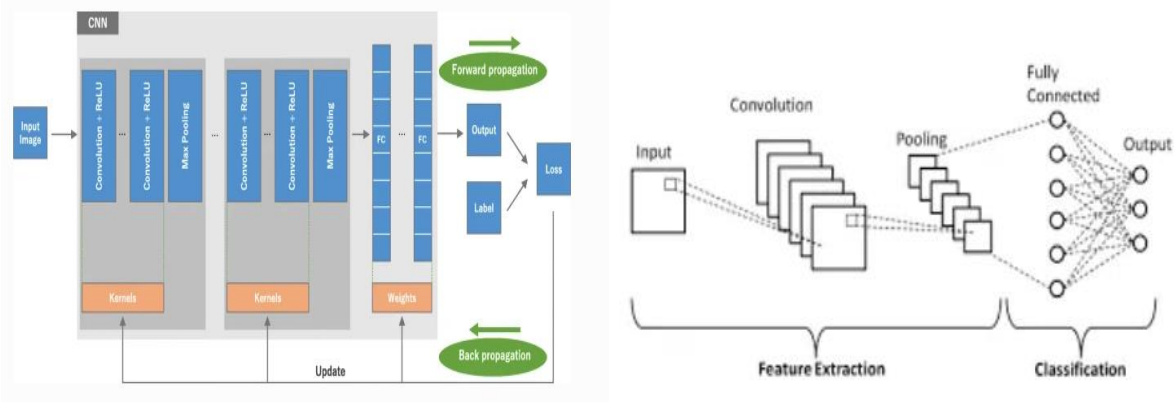

Figure 8A: Overview of the CNN Architecture (Yamashita, 2018)

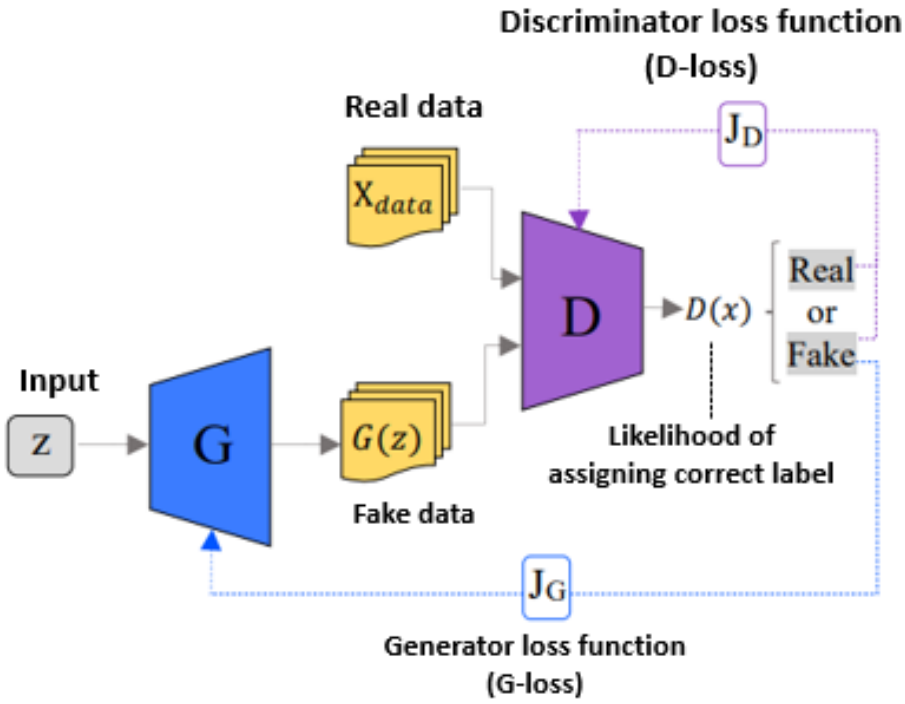

Figure 9A: The architecture of GAN (Salehi et. al., 2020)

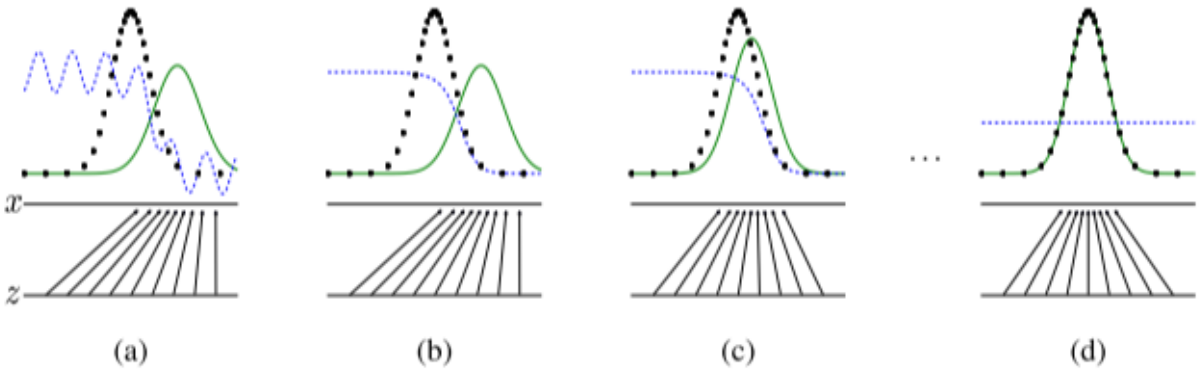

Figure 10A: Overall training process of GAN framework

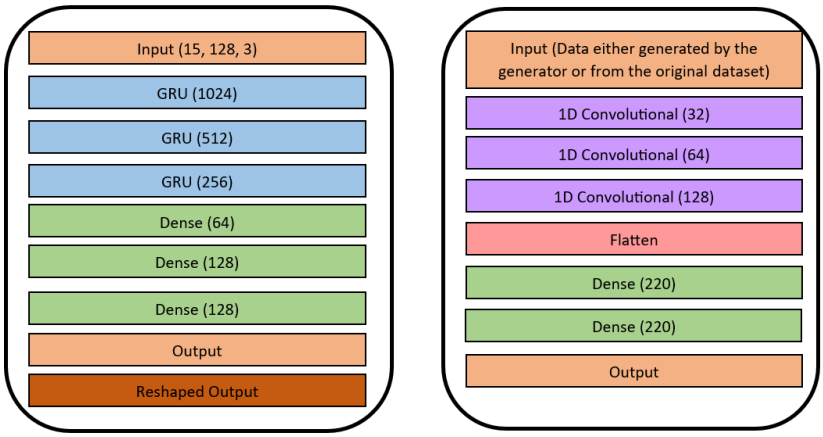

Figure 11A: Architecture of the generator and discriminator

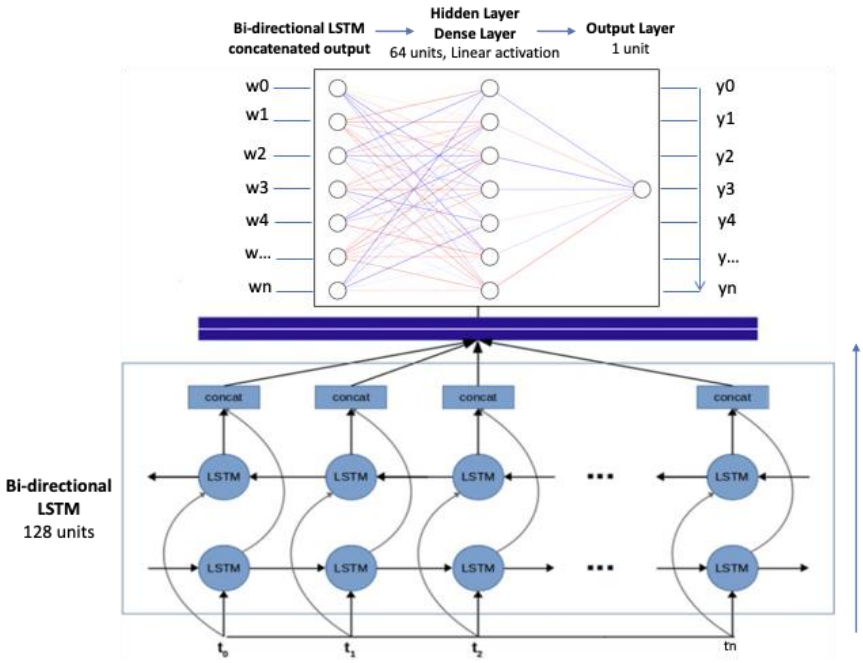

Figure 12A: Architecture of LSTM model

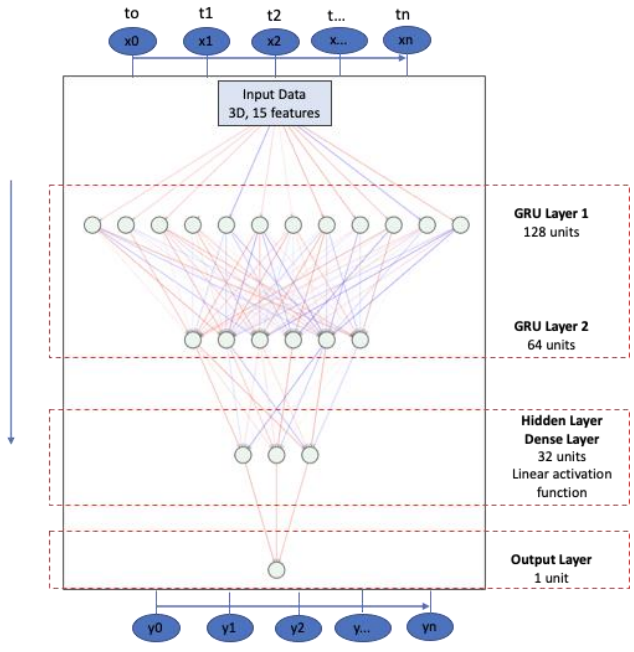

Figure 13A: Architecture of GRU model

B.) Tables

No.	Feature type	Feature	Data source
1.)	Basic feature	Closing price	yfinance library
2.)	Technical indicators	SMA (21 day)	Calculated from closing price
3.)		RSI (14-day window length)	
4.)		MACD (12 short, 26 long)	
5.)		Upper Bollinger band	
6.)		Lower Bollinger band	
7.)	Fundamental indicators	Earnings per share / closing price	WRDS
8.)		Book value per share / closing price	
9.)		Debt-to-equity ratio / closing price	
10.)		Net profit margin / closing price	
11.)		Federal interest rate	yfinance library
12.)		Gold price	
13.)		Crude oil price	
14.)		S&P500 index	
15.)		Nasdaq100 index	

Table 1A : The overview of variables used as inputs for deep learning models

No.	Technical Indicators	Formulas
1.)	Moving Average Convergence Divergence (MACD)	$MACD = EMA_{12}(p) - EMA_{26}(p)$ $where\ EMA = Exponential\ Moving\ Average$
2.)	Relative Strength Index	$RSI = 100 - \frac{100}{1 + RS}\ where\ RS = \frac{Average\ Gain}{Average\ Loss}$
3.)	21-day Simple Moving Average	$SMA = \frac{X_1 + X_2 + \ldots + X_n}{n}$ $where\ X_n = price\ of\ stock\ at\ period\ n\ and$ $n = number\ of\ total\ periods = 21$
4.)	Bollinger Upper Band	$BB_{upper} = x_i + (\sigma_i\ x\ d)$ $where:$ $x_i = SMA\ 20$ $\sigma_i = Standard\ Deviation\ of\ SMA\ 20\ and$ $d = deviations\ away\ from\ mean = 2$
5.)	Bollinger Lower Band	$BB_{lower} = x_i - (\sigma_i * d)$ $where:$ $x_i = SMA\ 20$ $\sigma_i = Standard\ Deviation\ of\ SMA\ 20\ and$ $d = deviations\ away\ from\ mean = 2$

Table 2A : Formulas for technical indicators used as features for deep learning models