Fast Genetic Algorithm for Long Short-Term Memory Optimization

Abba Suganda Girsang and Daniel Tanjung

Abstract—**This research aims to propose a version of Fast Genetic Algorithm (FGA), namely Fitness Value Memoization Genetic Algorithm (FVMGA). FVMGA uses the concept of memoization to cache the fitness value of chromosomes that have already been calculated before. It allows FVMGA to bypass unnecessary computation for redundant chromosome configurations, which is especially important when we use expensive fitness functions. For benchmarking purposes, the proposed FVMGA was compared to a traditional GA in the use case of optimizing Long Short-Term Memory (LSTM) hyperparameters for time-series forecasting. Four hyperparameters were being optimized in this study with a total of 38,000 possible combinations. However, the number was drastically reduced to 1,000 with the use of GA. The final results showed that FVMGA was able to compute up to 291% faster than traditional GA while maintaining the quality of the produced models.**

Index Terms—**genetic algorithm, hyperparameters, long short-term memory, memoization**

I. Introduction

IT is impressive to see how much computational power has grown to this day. It is just like what Gordon E. Moore, the co-founder of Intel, had stated in 1965, that the number of capacitors in a silicon chip would double every two years [1]. This statement, known as Moore's Law, had been proven correct for about 50 years. Although Moore's Law had come to an end in the last few years, it tells just how tremendous the growth of computational power is [2].

The significant increase in computational power brings new possibilities. [3] noted that even in the 1980s, people had known about the usefulness of neural networks and machine learning. However, at that time, computers were very slow. Many computations were too costly to be executed. Now that computational power is abundant, many complex techniques that were not possible in the 20th century have become accessible by everyone. Practical techniques like machine learning [4]–[9] and evolutionary algorithms [10]–[14] are rising in popularity.

It is understandable why machine learning may not be that popular decades ago as computational power and data used to be scarce. Nowadays, we could find data everywhere since everyone connects to the internet. However, not every record in raw data is valuable.

Manuscript received January 17, 2021; revised November 22, 2021.
Abba Suganda Girsang is a lecturer at Computer Science Department, BINUS Graduate Program-Master of Computer Science, Bina Nusantara University, Jakarta, Indonesia 11480 (corresponding author, e-mail: agirsang@binus.edu).
Daniel Tanjung is a master student at Computer Science Department, BINUS Graduate Program-Master of Computer Science, Bina Nusantara University, Jakarta, Indonesia 11480 (e-mail: daniel.tanjung@binus.ac.id).

Usually, there are many useless pieces of information contained in the raw data. Thus, we need a tool that can effectively extract useful information from the dump of data. Machine learning was introduced to allow the computer to learn from data and deduce crucial information with minimal human intervention [15].

One of the most popular implementations of machine learning is time-series forecasting. In the last decade, stock trading is rising in popularity. However, the stock price is a complex time-series with high volatility and is very unpredictable [16]. Small things like daily news could easily sway people's sentiments [17], [18]. Studies found that fear and anger very much affect people's decision-making ability [19], [20]. That is why people invest time and resources to build a reliable machine learning model to predict the stock market's future values.

However, when creating a machine learning model, a set of hyperparameters need to be defined before the training begins. These hyperparameters are variables that need to be assigned a value manually. They could significantly affect the performance of the models being produced. Therefore, searching for the optimal hyperparameters configuration has been a task with significant importance to build great quality models [21]

For the longest time, hyperparameter optimization or hyperparameter tuning was done manually [22]. Manually assigning a value for the hyperparameters is not an easy task for non-experts. However, this problem could be addressed by trying many combinations within a specified range. On the other hand, trying out many combinations would mean higher time and resource complexity.

Alternatively, heuristic search algorithms like Genetic Algorithm (GA) could be used to search the optimal hyperparameters configuration [23]–[26]. The purpose behind heuristic searching algorithms' existence is to find an optimal solution for very complex problems that are not possible or too expensive to solve using classical methods.

Fast Genetic Algorithm (FGA) is a term people used to describe several modified GA versions that compute faster than GA [27]–[30]. So, FGA is not a name designated for a specific algorithm. Each one of FGA has a different approach on how they achieve a faster computation. However, since modifications were made, FGA usually has some limitations and could only be used for specific optimization problems.

In this work, the author proposes a version of FGA that could effectively deal with time-consuming fitness functions like hyperparameter optimization. Sometimes, there are duplicate entries of the chromosome when optimizing with GA. It might not be a big problem if the fitness function does not have an expensive time complexity. However, when tuning hyperparameters, duplicate chromosomes lead to a

significant waste of time and computing resources as the fitness function is usually costly.

The proposed FGA, Fitness Value Memoization Genetic Algorithm (FVMGA), uses the concept of *memoization* to bypass the need to re-evaluate redundant chromosome configurations, thus effectively reduces the time complexity. Memoization is a type of caching method. It is not a strange concept in the Computer Science field. However, there has not been an attempt to integrate this clever concept into GA.

II. Literature Review

A. Related Works

Hyperparameter optimization is a critical subject in machine learning [21]. In the past, the task used to be done manually [22]. However, nowadays, hyperparameter optimization is done using automated alternatives, like heuristic search algorithms [23]–[26], [31].

GA is one of the most popular choices of heuristic search algorithms that people use to optimize hyperparameters. An experiment was carried out to compare Grid Search (GS), Random Search (RS), and GA for building Convolutional Neural Networks (CNN) [25]. It was shown that GA produced a great performing model with 86% accuracy. Meanwhile, GS falls behind the other two with only 83% accuracy.

In the field of Natural Language Processing (NLP), [24] experimented with using GA for optimizing hyperparameters of Artificial Neural Networks (ANNs). The results proved that using heuristic search algorithms like GA for hyperparameters optimization is very effective. The optimized hyperparameters gave better results than following the standard and recommended values that other people suggested online.

[23] used GA to optimize the hyperparameters of CNN with the use case of gravitational wave classification. Comparisons were made between GA and Deep Filtering that was introduced in [32]. It was shown that the GA-optimized model has 79% fewer trainable parameters while its classification accuracy is 11% higher compared to the Deep Filtering model.

[26] made a summarization of hyperparameter tuning techniques: GS, RS, Gradient-based models, Hyperband, Bayesian Optimized variations, GA, and Particle Swarm Optimization (PSO). It was described in detail that although GA is not the fastest algorithm, it is flexible and efficient for all types of hyperparameters. It also does not require careful initialization, unlike some other algorithms where careful initialization is needed.

Researchers had proposed variants of FGA to optimize GA's time complexity. [29] intended to cut computation time by finding patterns in the chromosomes' genes. Occasionally, every chromosome's genes will be checked, starting from the first index until the last index. If an index of everybody's chromosome all have the same value, that index becomes fixed. That chromosome index will not be included in future calculations. This action eliminates portions of the solution space and could make the algorithm converge faster. The proposed algorithm, Pattern Reduction Enhanced Genetic Algorithm (PREGA), was tested on Travelling Salesman Problem (TSP) ranging from 574 to 2,152 cities. PREGA achieved faster computation time from 28% up to 84% compared to traditional GA.

[28] also proposed a version of FGA that could converge faster. The idea behind it has to do with the initialization of the population at the algorithm's start. While the standard GA uses random initialization for its chromosomes, the proposed algorithm uses a new Small Region Creation Method (SRCM) method. SRCM divides the initial population evenly across the solution space. Another notable modification is the use of immune stochastic tournament selection. It adds the concept of thickness into the selection operator, which was inspired by the stimulative reaction and restraining reaction in the immune system. The combined use of both methods is believed can avoid premature convergence of the algorithm. The results showed that the proposed FGA indeed leads to faster convergence compared to standard GA.

[30] took another approach in modifying GA to suit a specific field of Orthogonal Frequency Division Multiplexing (OFDM) based Cognitive Radio System. GA was used to min-max radio channels' usage between Primary Users (PU) and Secondary Users (SU). The proposed GA removes the crossover phase of traditional GA. Instead, it introduces a repair phase that can repair chromosomes that did not meet the selection criteria. At the end of the study, the performance of the FGA was compared to two other algorithms, Minimum-Interference Algorithm (MIA) and Minimum-Power Algorithm (MPA). It was showed that the proposed FGA does a better job than MIA and MPA.

Another proposition is to modify the mutation operator of the algorithm [27]. Instead of using the usual recommended mutation rate of p=1/n where n is the length of the chromosome, it is changed into a random rate of p=α/n. The value of α is chosen from a random number $\alpha \in [1 \ldots n/2]$ according to a power-law distribution ${D_n}^{-\beta}$, where $\beta > 1$. After obtaining the value of α, it is just the usual mutation with the rate of p=α/n. It was also found that smaller β values like 1.5 produce better results than 2, 3, 4, and so on. This proposition of using a heavy-tailed mutation operator proved to be faster at finding global optimal compared to the one using the standard mutation rate.

The author himself has another take on how to model an FGA. The author wants to propose an FGA that suits optimization problems with high time complexity, such as optimizing machine learning models. The author realized that sometimes when optimizing using GA, there are duplicate chromosomes. These duplicate chromosomes are re-evaluated every time they appear. The author proposed Fitness Value Memoization Genetic Algorithm (FVMGA) as a solution that can bypass the need to re-evaluate duplicate chromosomes. Using the concept of memoization, we can cache previously calculated individuals along with their fitness values. The concept is straightforward but effective in achieving its purpose.

B. Time-Series

Time-series is a *sequence* of data that is sorted by date. In most cases, the data or records are recorded at a fixed interval, like hourly, daily, weekly, monthly, yearly, and so forth. When drawn in a graph, the x-axis would serve as the date. In contrast, its y-axis represents the observed variables' value or any quantitative variables throughout the recording time. Fig.

1 shows an example of a time-series when drawn in a line graph.

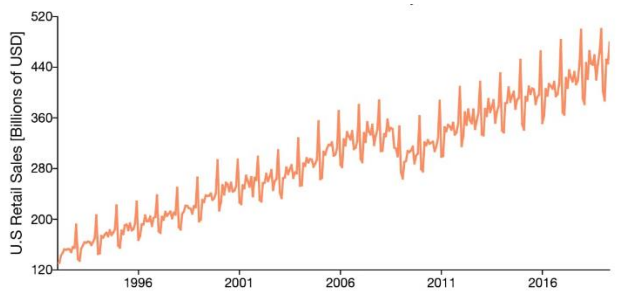

Fig. 1. An example of a time-series.

In every time-series, there are vital components that data analysts usually use to analyze the data. These key components are the *trend* and *seasonality* of the data. Both trend and seasonality can be found in many real case time-series. They make up a large proportion of time-series themselves. They can even be pretty much considered as the time-series itself.

Every time-series in the world will have a trend. However, some time-series may not have seasonality. While both components are crucial, data analysts focus only on the trend in most cases. This is because, often, the seasonality of a time-series will hardly change at all. Thus, even people who are not experts could analyze the seasonality of a time-series with ease. The representation of trend and seasonality of time-series can be seen in Fig. 2.

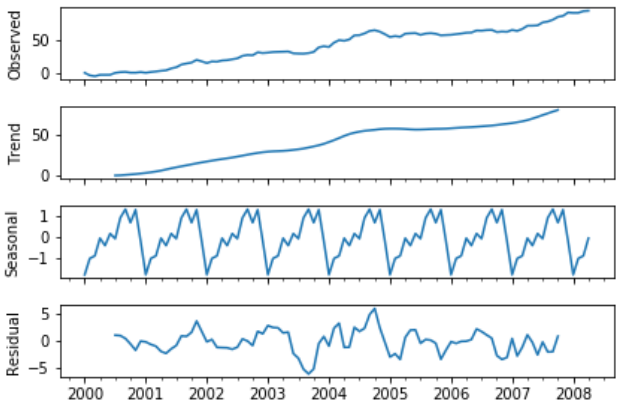

Fig. 2. An example of a time-series decomposition.

As shown in Fig. 2, the trend of time-series describes the change in the observed variables' value within the range of observation. A trend of a time-series could go upward, downward, or just flat/stable. There is also another component called the *residual*. Residual/Error is a component within time-series that is neither trend nor seasonality. This component is uncontrollable and erratic and can be considered as interruptions from external factors, such as nature. Also, in Fig. 2, there are grey rectangles on the right side of the four components. Those are the scale of the height of the components relative to each other. So, it does not mean that the trend is curvy, but its height was scaled, making it looks like that. The truth is, if every component uses the same scale, then the trend component of that time-series would look more like a straight line rather than curves.

Another critical part of building a model is to choose a suitable range of records. Most data scientists like to use a time-series with a larger number of records or data rows. This is because the time-series would be most likely to be reliable. Time-series with a low number of records will most likely produce unreliable results. However, this does not mean that a larger number of records will always produce better results. Sometimes, time-series with many records can produce worse results than time-series with a low number of records. It is the job of data analysts to decide the ideal range of records to design a model.

C. Time-Series Forecasting

Time-series forecasting is a field where people try to predict future values of time-series data. Forecasting has always been highly regarded in many fields, including business, economics, government, politics, and environmental science [33]. As the name suggests, a prediction is only a guess that is based on experience or knowledge. It will not have 100% accuracy. At best, it is only better than a wild guess. Nevertheless, making predictions helps in building a degree of confidence. It also helps in planning and the making of critical decisions [34].

A vital part of time-series forecasting is to choose the proper range of records. Most people like to use a time-series with many records because time-series with a small number of records are more likely to produce unreliable results. However, a larger number of records is not necessarily better. Almost every dataset in the world has some noise and unwanted or low-quality records that could negatively impact the forecast. Besides deciding the range of records, filtering out the noise and low-quality records is also essential in time-series forecasting.

D. Long Short-Term Memory

Long Short-Term Memory (LSTM) is a part of machine learning within the scope of neural networks. Because it is a neural network, LSTM has an exceptional adaptation ability to any problems. LSTM was made as a modification to Recurrent Neural Network (RNN). [35] introduced LSTM as an alternative to RNN, which has the problems of vanishing gradient and exploding gradient. Both vanishing and exploding gradients negatively affect the model. They hinder the model from learning effectively from distant information of the past. [36], [37] discussed RNN's problems of vanishing and exploding gradients in detail.

As an improved version of RNN, LSTM introduced three additional gates for input to pass through: forget gate, input gate, and output gate. The three gates help in regulating the long-term memory state of the LSTM. The forget gate decides which information will be removed from the cell state that we got from the previous timestep. Then input gate decides what new information from the current timestep that we would want to store into the cell state. Lastly, the output gate will filter the information and decide which information will be passed to the next timestep.

As a part of deep learning, LSTM has advantages over classical methods such as Auto-Regressive Integrated Moving Average (ARIMA) in its flexibility. Meanwhile, ARIMA, as one of the classical forecasting methods, is limited to simple time-series. When forecasting complex time-series like stock prices, classical methods like ARIMA lose to neural networks like LSTM [9], [38].

E. Hyperparameter Optimization

Hyperparameter optimization or hyperparameter tuning is critical in building the best neural network models [21]. Many machine learning models have a set of hyperparameters that need to be appropriately configured before training. These hyperparameters could affect the final model's quality. So, choosing the correct values for them is a vital task, although it is not easy.

Hyperparameter tuning used to be a manual [22]. However, even experts sometimes have a hard time tuning hyperparameters. There are several methods that computers could help us. One of the options computers could help us optimize hyperparameters is using heuristic search algorithms [23]–[26].

F. Genetic Algorithm

Genetic Algorithm (GA) has always been popular and widely used ever since the 1980s [10], [11]. The idea behind it was taken from Charles Darwin's theory of *survival of the fittest*. It is only expected that the fittest of all would survive to the next generation in natural evolution. This theory of natural evolution is easy to understand and is implemented in GA. Fig. 3 describes how elements are represented in GA.

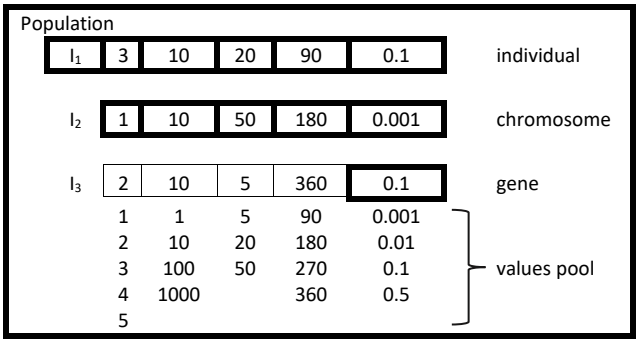

Fig. 3. GA representation.

In GA, there is a universe of discourse called *population*. Within a population, several *individuals* are living in it. Each individual has a sequence of a *chromosome*. Everybody's chromosome may contain several *genes*. Then, each index of genes has its own values pool according to its index. A chromosome is also often called a *candidate solution*.

Each candidate solution has a *fitness value*. The fitness value could be found as an output from inputting the candidate solution to a *fitness function*. There should only be one fitness function that can process every candidate solution within the population for one problem. The calculation made within the function could be anything. The overall flowchart of GA is shown in Fig. 4.

After obtaining the fitness value of each candidate solution, then it goes to the selection phase. In the *selection phase*, the fittest individual will be chosen to inherit offspring to the next generation. In the *crossover phase*, two chromosomes from two different individuals are used. Some of the genes will be swapped between the two chromosomes, resulting in two new individuals. The two new individuals would then enter the mutation phase.

Within the *mutation phase*, the two individuals will experience mutation in some of their genes. A mutation changes the said gene's value to another value within the said index's values pool. Since there are two new individuals, the current population has exceeded the limit, and two individuals must leave the population. The technique to choose who leaves and who stays may vary depending on the model. Finally, check if the algorithm should stop or continue to loop to the selection phase.

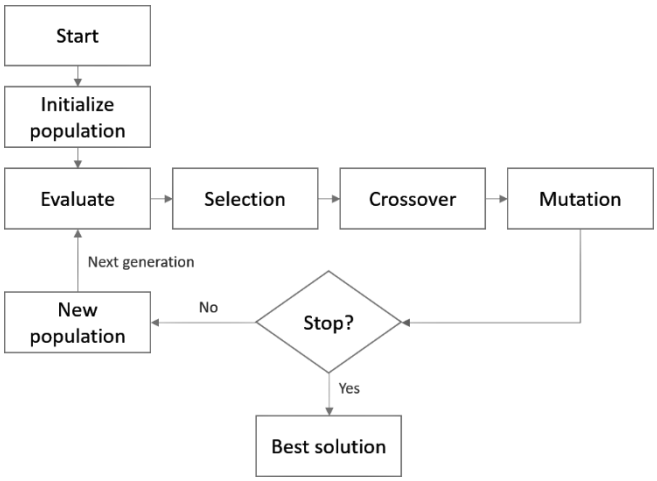

Fig. 4. The flowchart of GA.

The first important thing to do when using GA would be to define its structure. For example, the number of the gene, the values pool, the number of individuals within a population, how many would be selected as parents, the method of parent selection, the method of crossover, the method of mutation, etc. After that, the flow of a traditional Genetic Algorithm would be like this:

1. Calculate the population's fitness.
2. *Selection Phase*: select individuals from within the population to be parents.
3. *Crossover Phase*: swap genes between 2 parents to produce children.
4. *Mutation Phase*: change the value of some gene at the given probability.
5. Repeat from (1) until it satisfies the stop criteria.

III. Proposed Method

The core idea of the algorithm is to remove redundancy from the Genetic Algorithm process. Usually, GA needs to re-evaluate each duplicate chromosome configuration because it does not recognize them. This issue is what the author wants to address by introducing FVMGA. Note that in Fig. 5, the same chromosome configuration was trained twice on two different LSTM models. For example, if training one LSTM model takes 160 seconds, re-evaluate it the second time would mean wasting another 160 seconds.

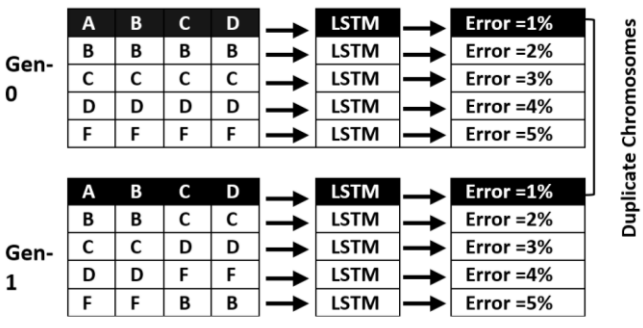

Fig. 5. Traditional GA re-evaluates duplicate chromosomes.

Re-evaluating duplicate chromosomes would not be a big problem when the fitness function could be executed in a concise amount of time. However, even a simple LSTM takes time to train, not to mention one with a complex structure with more layers. Fig. 6 shows how the *memoization* concept could cut time when dealing with redundancy.

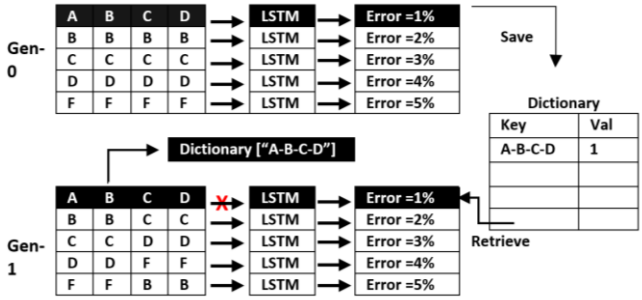

Fig. 6. FVMGA bypasses duplicate chromosomes.

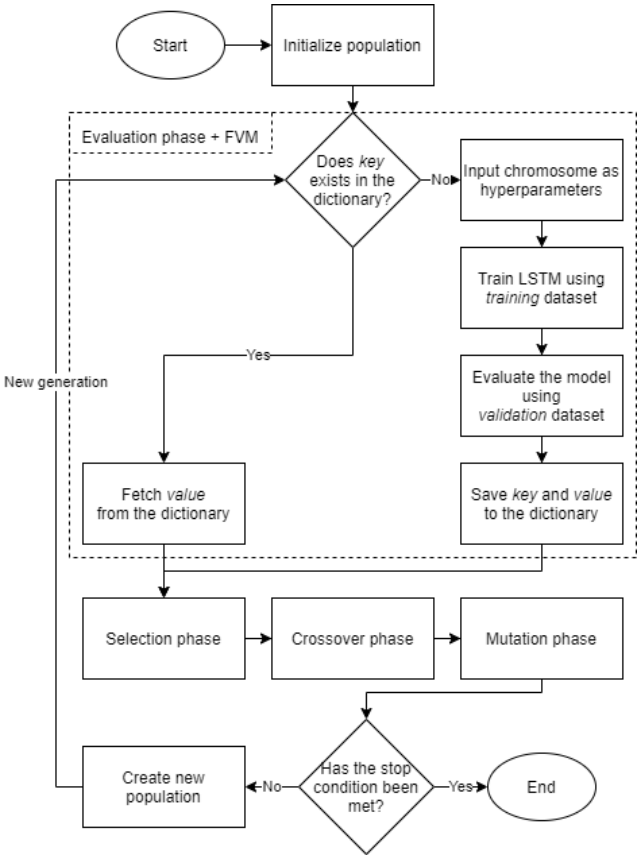

Fig. 7. The flowchart of FVMGA-LSTM.

Fig. 7 describes the flowchart of FVMGA for the use case of optimizing LSTM hyperparameters. The *evaluation phase* is the only difference that FVMGA has over GA. Instead of directly evaluating every chromosome, an additional algorithm searches whether the chromosome is a duplicate. The duplicate chromosomes will not be re-evaluated, thanks to the *memoization* concept.

Memoization is one type of *caching* technique that stores up results from expensive function calls [39]. Memoization could bypass wasteful redundant computation because it remembers results from previous operations. We implement the memoization concept to enable GA to remember every chromosome that had been processed, along with their fitness value. So, when a duplicate appears, the algorithm could skip the fitness function entirely. It only needs to look up the memo dictionary to find the corresponding chromosome's fitness value. The simplified flow of FVMGA is:

1. Evaluate the population's fitness with the help of the fitness value memoization (FVM) algorithm.
2. Selection phase
3. Crossover phase
4. Mutation phase
5. Repeat from (1) until it satisfies stop criteria.

The only difference between FVMGA and GA lies in the evaluation phase (i.e., step 1). In the evaluation phase of FVMGA, there is an additional algorithm called fitness value memoization (FVM). It keeps track of which chromosomes had been calculated before and their fitness value. The flow of the FVM algorithm is:

1. Generate a key based on each chromosome configuration.
2. If the key exists in the dictionary, go to (5); otherwise, go to (3).
3. Calculate the fitness value of the given chromosome configuration using the fitness function.
4. Insert the key into the dictionary and set its value to what was returned from the fitness function.
5. Return the value of the key.

IV. Experimental Results

A. Dataset

Several datasets were used in this research to have a better quality of evaluation. All the datasets used in this work, including all the graphs presented in the previous sections of this thesis, were collected for free from various sources such as R language's dataset package, Kaggle, and Wikipedia. Table I listed the four datasets used in this research.

TABLE II
Dataset Used in the Experiment

Code	Description	Range	# of Rows
DS1	Daily records of the average temperature in Delhi, India	1 January 2013 – 24 April 2017	1575
DS2	Monthly records of the number of observed sunspots	1749 – 1983	2820
DS3	Daily records of the number of views of https://en.wikipedia.org/	1 July 2015 – 31 March 2020	1736
DS4	Daily records of the exchange rate of EUR-IDR	4 January 1999 – 20 April 2020	5451

The four datasets vary in context: DS1 and DS2 are about environmental science, DS3 is about business, and DS4 is about investment. Further details regarding each dataset could be seen in Table II.

TABLE II
Summary of Dataset

Code	Min	Max	Mean	Standard Deviation
DS1	6	38.714	25.231	7.337
DS2	0	253.8	51.265	43.448
DS3	2.028e+08	3.122e+08	2.524e+08	1.559e+07
DS4	6707.81	18239.61	12563.342	2680.091

B. Dataset Splitting

In this experiment, the datasets were split into three categories: training, validation, and testing. The training

dataset was used to train the LSTM models. Then, the fitness value of the LSTM models was evaluated using the validation dataset. Meanwhile, the testing dataset was not used in the optimization process. It was used later on after the optimization process was done to evaluate the eight final LSTM models (the best models that GA and FVMGA found for the four datasets). The splitting percentages for the datasets are shown in Fig. 8. The sizes of training, validation, and testing data are 80%, 10% and 10 %, respectively.

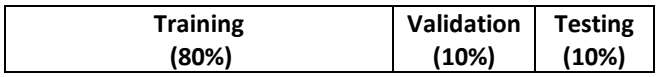

Fig. 8. Dataset splitting percentages

C. Chromosome Representation

In this study, we want to compare FVMGA to GA by using them to optimize LSTM hyperparameters. We decided to use four hyperparameters for this experiment. We used an array to represent each chromosome. So, each hyperparameter or gene represents an index within the array. Thus, each chromosome would contain four genes (i.e., the array has a length of four). Table III lists the four hyperparameters used in the experiment. The combination of the four hyperparameters results in 38,000 total possible combinations.

TABLE III
LIST OF OPTIMIZED HYPERPARAMETERS

Hyperparameter	Range Values	Explanation
Sliding window	10 to 100	Interval of 5
Dropout (d)	0.01 to 0.5	Interval of 0.01
LSTM units (u)	10 to 100	Interval of 10
Batch size	16, 32, 64, 128	-

Batch size is a crucial hyperparameter that would almost always be used in every neural network. It defines the number of samples that got through the network in one batch. For example, a dataset of 500 records was trained using a batch size of 128. That would mean that the algorithm will divide the dataset into four mini-batches with a length of 128, 128, 128, and 116. The first advantage of using mini-batches is that the training process requires less memory than using all of the samples all at once. The second advantage is that training using mini-batches is usually faster than using full-batch. However, training with a batch size that is too small can negatively impact the performance of the model because the training process becomes ineffective. So, choosing the suitable value for the batch size is critical.

A sliding window is another important hyperparameter in machine learning. It defines the size or window of past information that should be taken into account. For example, in time-series forecasting using LSTM, if we tell the LSTM to learn from today's temperature to predict tomorrow's temperature, it means that we use a sliding window of 1. However, sometimes we need more than one record of past information. For example, if we want to predict the traffic on a specific road, we might use the last entire week's information (i.e., the past seven days instead of just yesterday).

LSTM units are the number of hidden units within each LSTM cell. Deciding on how many LSTM units are used for each LSTM cell is also important. Using a larger number of LSTM units usually results in better training accuracy. However, the training process also becomes much slower. Also, using a number that is too big can result in over-fitting. Thus, we have to decide the number of LSTM units we use carefully.

A dropout is a regularization that randomly shuts down or excludes some inputs from going to the next layer. Adding dropout is a way to prevent the model from over-fitting. Dropout value is the percentage of how many inputs will be excluded (e.g., a dropout value of 0.2 means 20% of the inputs are shut down). However, a dropout value that is too high can hinder the model performance as the learning process become inefficient. Deciding the right amount of dropout value is essential in fighting over-fitting while maintaining the model's performance.

D. Parameters

With the use of GA, the complexity of the problem was reduced from 38,000 combinations to 1,000 combinations. These 1000 combinations are divided into 100 generations, with each generation having ten individuals. It was also decided that each chromosome will have four genes. These parameters and the other ones used for both LSTM and GA are listed in Table IV.

TABLE IV
LIST OF OTHER PARAMETERS

Parameter	Usage	Value
Epoch	LSTM	10
Loss	LSTM	Mean Squared Error
Optimizer	LSTM	Adam
Max generation	GA	100
Population size	GA	10
Number of gene (n)	GA	4
Elites	GA	2
Selection method	GA	Roulette
Crossover method	GA	Single point
Crossover rate	GA	0.7
Mutation rate	GA	1/4 (1/n)

There is one more crucial thing to note: LSTM uses some *randomization* within its architecture. So, training two LSTM models using the same parameters could yield different results. Of course, this is *not* a good thing when we want to optimize hyperparameters. To optimize the hyperparameters, we must evaluate the fitness of each chromosome using LSTM. However, the randomness of LSTM makes it hard to evaluate them correctly.

The author decided to train *multiple LSTM models* for every chromosome configuration to counter the problem. For each chromosome, we trained 100 models using the training dataset. We used the validation dataset for every model to evaluate them, resulting in 100 validation losses (i.e., error percentage). The error values from the LSTM model will then got converted to fitness value. Of course, this is 100 times more expensive in both time and resources. However, doing this helps in finding more consistency, although not perfect. We cannot correctly optimize the hyperparameters if each chromosome's fitness value continually changes.

E. Environment

The experiment was done using Python, as it is an excellent choice for machine learning activities. We used Keras library,

which makes building LSTM easier. CUDA technology from NVIDIA was also used to help speed up the training process of the LSTM models. Table V listed the environment settings used in this experiment.

TABLE V
ENVIRONMENT SETTINGS

Environment	Settings
CPU	Ryzen 3700X
GPU	NVIDIA GTX 970
CUDA	10.1
CUDNN	8.3
Python	3.7.7
Tensorflow	2.20
Keras	2.4.3

F. Evaluation

The FVMGA will be tested and compared against TGA using the case of optimizing LSTM for time-series forecasting. Several datasets were prepared in the hope of having a better quality analysis. The evaluation comprises a comparison in both time and quality aspects of both algorithms. We make sure to set the environment as fair as possible, using precisely the same machine and tools for both of them. The only difference between the two is that the FVMGA uses an additional algorithm to check each chromosome configuration and skip redundant or repeating ones.

G. Performance Metrics

As this work revolves around the Fast Genetic Algorithm, it only fits to say that the primary metric is the total computation time. FVMGA, as an FGA, is expected to be faster than traditional GA. Therefore, along with total computation time, we also provide the speed improvement percentage. However, although FVMGA has to be fast, it cannot compromise the quality of the models produced. After all, a heuristic search algorithm aims to find the optimal solution to problems. To compare the performance of the models that FVMGA and GA optimized, we used four performance metrics mentioned in Table VI.

TABLE VI
PERFORMANCE METRICS

Metric	Formula
Time (seconds)	$finish\ time - start\ time$
Improvement %	$\left(\frac{time_{FVMGA}}{time_{GA}} - 1\right) \times 100\%$
RMSE	$\sqrt{\frac{\sum_{i=1}^{n}(predicted_i - actual_i)^2}{n}}$
MAE	$\frac{\sum_{i=1}^{n}\lvert predicted_i - actual_i\rvert}{n}$
MAPE	$\frac{\sum_{i=1}^{n}\left\lvert\frac{predicted_i - actual_i}{actual_i}\right\rvert}{n}$
R^2 Score	$1 - \frac{\sum_{i=1}^{n}(predicted_i - actual_i)^2}{\sum_{i=1}^{n}(actual_i - mean\ of\ actual)^2}$

H. Experiment Result

FVMGA is proposed as a way to bypass duplicate chromosomes. When optimizing LSTM hyperparameters, it means that FVMGA could skip the need to train an LSTM model for duplicate chromosomes. Instead, FVMGA could retrieve the fitness value from the dictionary. So, we start the experiment by comparing the time it takes to train an LSTM model and look up the dictionary.

TABLE VII
TIME-COMPLEXITY COMPARISON BETWEEN TRAINING AN LSTM MODEL AND LOOKING UP THE DICTIONARY

Action	Average Execution Time (milliseconds)
Training an LSTM model	160,000.0
Looking up the dictionary	0.5

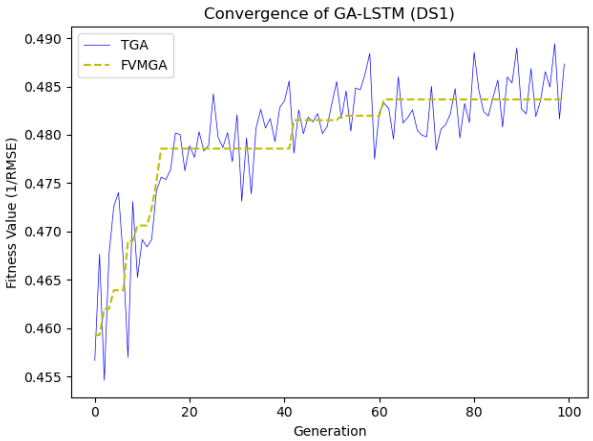

Fig. 9. Fitness Value Convergence of the Optimization Process (DS1).

The LSTM models were trained using the *training* dataset. They were then evaluated using the *validation* dataset, thus producing the *validation loss* as the fitness value. Fig. 9, 10, 11, and 12 show the best individual's fitness value over the 100 generations of the four datasets' optimization process. This experiment was done ten times, and the average time difference was described in Table VII.

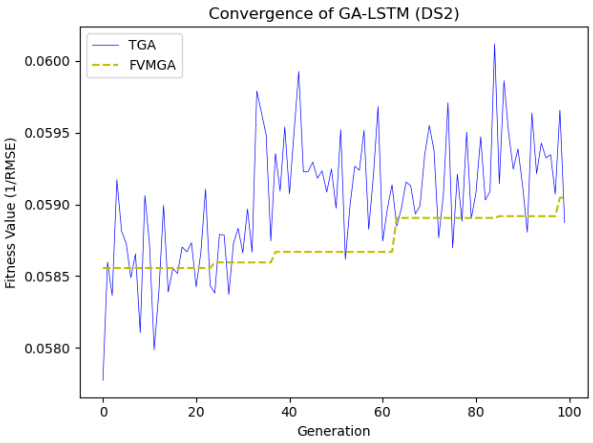

Fig. 10. Fitness Value Convergence of the Optimization Process (DS2).

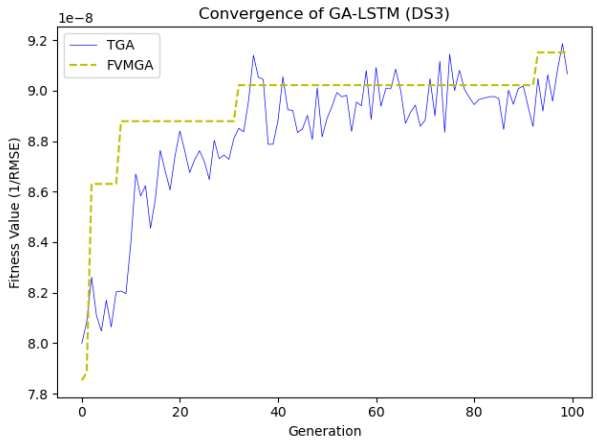

Fig. 11. Fitness Value Convergence of the Optimization Process (DS3).

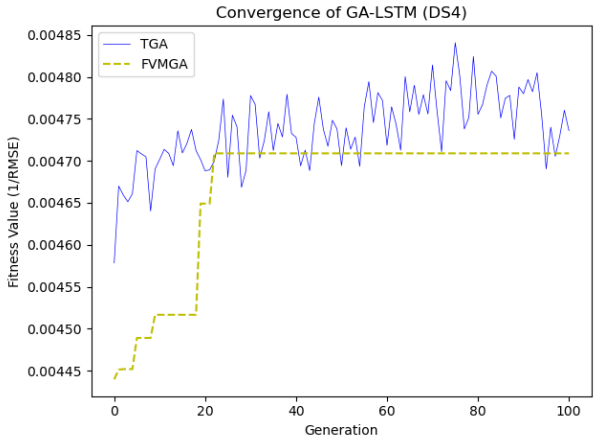

Fig. 12. Fitness Value Convergence of the Optimization Process (DS4).

The optimization process ended when the algorithms reached the 100th generation. The algorithms will then output the best-performing individual from the last generation. There are eight final LSTM models produced from the optimization process, two models for each dataset. One model was optimized using GA, and the other one was optimized using FVMGA.

For the final experiment, we must test eight LSTM models with the best chromosome configuration using the testing dataset. Remember that we have not used the testing dataset so far. That is because the testing dataset will be used to do the final testing for the eight models that had been optimized. Table VIII summarizes the final results of both GA-LSTM and FVMGA-LSTM for all datasets, with the better values being written in a bold font.

TABLE VIII
EXPERIMENTAL RESULTS

Code	Method	Performance/Quality			
		RMSE*	MAE*	MAPE*	R^{2**}
DS1	GA	1.744	1.402	**7.247**	**0.91692**
	FVMGA	**1.732**	1.402	7.285	0.91277
DS2	GA	**16.740**	**12.603**	**30.113**	**0.88346**
	FVMGA	16.814	12.689	31.088	0.88243
DS3	GA	8,761,847	6,856,679	2.64653	0.67246
	FVMGA	**8,700,089**	**6,792,297**	**2.62022**	**0.67714**
DS4	GA	174.995	128.254	1.57295	0.96098
	FVMGA	**170.965**	**122.391**	**1.50362**	**0.96390**

*lower value is better

**higher value is better

Fig. 13 shows that the difference in time-complexity between the two algorithms is huge. Table IX explores further the details regarding the significant improvement gained from using FVMGA.

Time is the most important point of comparison between traditional GA and FVMGA. After all, the most significant advantage that FVMGA has over GA is its speedy execution. Fig. 13 shows the total time that GA and FVMGA needed to optimize each model.

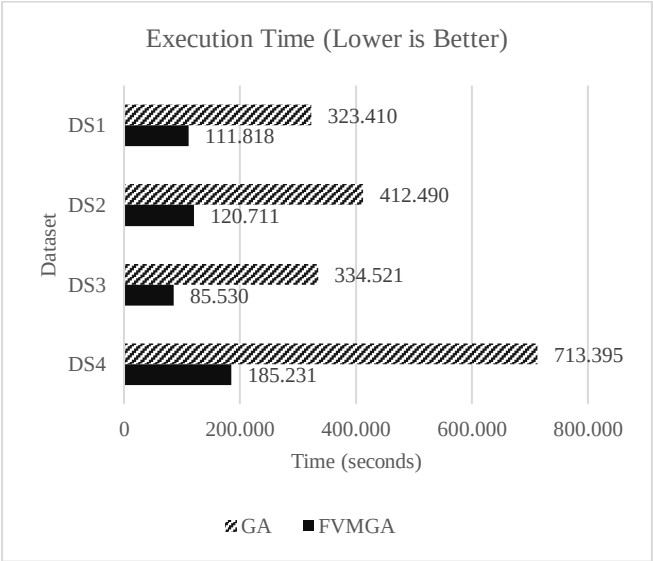

Fig. 13. The execution time of GA and FVMGA.

TABLE IX
SPEED IMPROVEMENT AND THE NUMBER OF DUPLICATE CHROMOSOMES

Code	Method	Time (seconds)	Improvement %	Duplicate Chromosomes (out of 1,000)
DS1	GA	323,410	0%	572
	FVMGA	**111,818**	**+189%**	560
DS2	GA	412,490	0%	530
	FVMGA	**120,711**	**+241%**	575
DS3	GA	334,521	0%	558
	FVMGA	**85,530**	**+291%**	594
DS4	GA	713,395	0%	599
	FVMGA	**185,231**	**+285%**	602

Table IX shows that around 500-600 out of 1,000 chromosomes were duplicates (i.e., around 50% to 60%). That was a significant number of duplicate chromosomes. The model produced by FVMGA is slightly 291% better than the original of method GA. From the results obtained, it is clear that the proposed algorithm, Fitness Value Memoization Genetic Algorithm (FVMGA), is a success and could be considered as one of Fast Genetic Algorithm (FGA).

V. CONCLUSION

Often, in the process of finding the optimal solution, GA has to deal with duplicate chromosomes. A traditional GA cannot recognize whether a chromosome is a duplicate or not. GA will constantly naively re-evaluate those duplicate chromosomes' fitness value. Depending on how complex the fitness function is, this could lead to a considerable waste of

time. However, this problem could be addressed by caching every chromosome that had been evaluated before.

FVMGA uses the concept of memoization to cache the results from expensive fitness functions. Before we send the chromosome to the fitness function, we first check whether the chromosome is a duplicate or not. If it is, then we only need to fetch the cached value. If it is not a duplicate, then we evaluate the chromosome using the fitness function as usual.

REFERENCES

[1] G. E. Moore, "Lithography and the future of Moore's law," in *Integrated Circuit Metrology, Inspection, and Process Control IX*, 1995, vol. 2439, p. 2, doi: 10.1117/12.209195.

[2] M. M. Waldrop, "More than Moore," *Nature*, vol. 530, no. 7589. pp. 144–147, 2016, doi: 10.1038/530144a.

[3] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science (80-.).*, vol. 313, no. 5786, pp. 504–507, 2006, doi: 10.1126/science.1127647.

[4] A. A. Adebiyi, A. O. Adewumi, and C. K. Ayo, "Comparison of ARIMA and Artificial Neural Networks models for stock price prediction," *J. Appl. Math.*, vol. 2014, 2014, doi: 10.1155/2014/614342.

[5] K. Chen, Y. Zhou, and F. Dai, "A LSTM-based method for stock returns prediction: A case study of China stock market," in *Proceedings - 2015 IEEE International Conference on Big Data, IEEE Big Data 2015*, 2015, pp. 2823–2824, doi: 10.1109/BigData.2015.7364089.

[6] Y. Kara, M. Acar Boyacioglu, and Ö. K. Baykan, "Predicting direction of stock price index movement using Artificial Neural Networks and Support Vector Machines: The sample of the Istanbul Stock Exchange," *Expert Syst. Appl.*, vol. 38, no. 5, pp. 5311–5319, 2011, doi: 10.1016/j.eswa.2010.10.027.

[7] R. Fu, Z. Zhang, and L. Li, "Using LSTM and GRU neural network methods for traffic flow prediction," in *Proceedings - 2016 31st Youth Academic Annual Conference of Chinese Association of Automation, YAC 2016*, 2017, pp. 324–328, doi: 10.1109/YAC.2016.7804912.

[8] Z. C. Lipton, D. C. Kale, C. Elkan, and R. Wetzel, "Learning to diagnose with LSTM recurrent neural networks," in *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, 2016.

[9] S. McNally, J. Roche, and S. Caton, "Predicting the price of Bitcoin using machine learning," in *Proceedings - 26th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2018*, 2018, pp. 339–343, doi: 10.1109/PDP2018.2018.00060.

[10] J. H. Holland, *Adaptation in natural and artificial systems*. 2019.

[11] M. Mitchell, *An introduction to Genetic Algorithms*, vol. 32, no. 6. 1996.

[12] V. Roberge, M. Tarbouchi, and G. Labonte, "Comparison of parallel Genetic Algorithm and Particle Swarm Optimization for real-time UAV path planning," *IEEE Trans. Ind. Informatics*, vol. 9, no. 1, pp. 132–141, 2013, doi: 10.1109/TII.2012.2198665.

[13] B. Saeidian, M. S. Mesgari, and M. Ghodousi, "Evaluation and comparison of Genetic Algorithm and Bees Algorithm for location-allocation of earthquake relief centers," *Int. J. Disaster Risk Reduct.*, vol. 15, pp. 94–107, 2016, doi: 10.1016/j.ijdrr.2016.01.002.

[14] G. A. E.-N. A. Said, A. M. Mahmoud, and E.-S. M. El-Horbaty, "A Comparative Study of Meta-heuristic Algorithms for Solving Quadratic Assignment Problem," *Int. J. Adv. Comput. Sci. Appl.*, vol. 5, no. 1, 2014, doi: 10.14569/ijacsa.2014.050101.

[15] A. T. Kolokouris, "Machine learning," *Byte*, vol. 11, no. 12, pp. 225–226, 228, 1986, doi: 10.4018/ij3dim.2017070101.

[16] Y.-F. Huang and R. Startz, "Improved Recession Forecasts considering stock market volatility," *SSRN Electron. J.*, 2018, doi: 10.2139/ssrn.3297949.

[17] B. J. Vanstone, A. Gepp, and G. Harris, "Do news and sentiment play a role in stock price prediction?," *Appl. Intell.*, vol. 49, no. 11, pp. 3815–3820, 2019, doi: 10.1007/s10489-019-01458-9.

[18] B. J. Vanstone, A. Gepp, and G. Harris, "The effect of sentiment on stock price prediction," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018, vol. 10868 LNAI, pp. 551–559, doi: 10.1007/978-3-319-92058-0_53.

[19] E. Gilbert and K. Karahalios, "Widespread worry and the stock market," in *ICWSM 2010 - Proceedings of the 4th International AAAI Conference on Weblogs and Social Media*, 2010, pp. 58–65.

[20] J. S. Lerner and D. Keltner, "Fear, anger, and risk," *J. Pers. Soc. Psychol.*, vol. 81, no. 1, pp. 146–159, 2001, doi: 10.1037/0022-3514.81.1.146.

[21] M. Claesen and B. De Moor, "Hyperparameter search in machine learning," pp. 10–14, 2015, [Online]. Available: http://arxiv.org/abs/1502.02127.

[22] R. Bardenet, M. Brendel, B. Kégl, and M. Sebag, "Collaborative hyperparameter tuning," in *30th International Conference on Machine Learning, ICML 2013*, 2013, no. PART 2, pp. 858–866.

[23] D. S. Deighan, S. E. Field, C. D. Capano, and G. Khanna, "Genetic-Algorithm-optimized neural networks for gravitational wave classification," *arXiv*. 2020.

[24] N. Gorgolis, I. Hatzilygeroudis, Z. Istenes, and L. N. G. Gyenne, "Hyperparameter optimization of LSTM network models through Genetic Algorithm," in *10th International Conference on Information, Intelligence, Systems and Applications, IISA 2019*, 2019, doi: 10.1109/IISA.2019.8900675.

[25] P. Liashchynskyi and P. Liashchynskyi, "Grid Search, Random Search, Genetic Algorithm: A big comparison for NAS," *arXiv*. 2019.

[26] L. Yang and A. Shami, "On hyperparameter optimization of machine learning algorithms: Theory and practice," *Neurocomputing*, vol. 415, pp. 295–316, 2020, doi: 10.1016/j.neucom.2020.07.061.

[27] B. Doerr, R. Makhmara, H. P. Le, and T. D. Nguyen, "Fast Genetic Algorithms," in *GECCO 2017 - Proceedings of the 2017 Genetic and Evolutionary Computation Conference*, 2017, pp. 777–784, doi: 10.1145/3071178.3071301.

[28] W. Gao, "Study on improved Fast Immunized Genetic Algorithm," in *Proceedings - 2nd International Conference on Genetic and Evolutionary Computing, WGEC 2008*, 2008, pp. 55–58, doi: 10.1109/WGEC.2008.67.

[29] S. P. Tseng, C. W. Tsai, M. C. Chiang, and C. S. Yang, "Fast Genetic Algorithm based on pattern reduction," in *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, 2008, pp. 214–219, doi: 10.1109/ICSMC.2008.4811277.

[30] M. Yuan, S. Wang, and S. Du, "Fast Genetic Algorithm for bits allocation in OFDM based Cognitive Radio Systems," in *WOCC2010 Technical Program - The 19th Annual Wireless and Optical Communications Conference: Converging Communications Around the Pacific*, 2010, doi: 10.1109/WOCC.2010.5510609.

[31] J. T. Tsai, J. H. Chou, and T. K. Liu, "Tuning the structure and parameters of a neural network by using hybrid Taguchi-Genetic Algorithm," *IEEE Trans. Neural Networks*, vol. 17, no. 1, pp. 69–80, 2006, doi: 10.1109/TNN.2005.860885.

[32] D. George and E. A. Huerta, "Deep neural networks to enable real-time multimessenger astrophysics," *arXiv*. 2016.

[33] L. Seymour, P. J. Brockwell, and R. A. Davis, "Introduction to time series and forecasting.," *J. Am. Stat. Assoc.*, vol. 92, no. 440, p. 1647, 1997, doi: 10.2307/2965440.

[34] D. C. Montgomery, C. L. Jennings, and M. Kulahci, *Introduction to time series analysis and forecasting*. Hoboken: John Wiley & Sons, Inc., 2008.

[35] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997, doi: 10.1162/neco.1997.9.8.1735.

[36] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with Gradient Descent is difficult," *IEEE Trans. Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994, doi: 10.1109/72.279181.

[37] J. F. Kolen and S. C. Kremer, "Gradient Flow in Recurrent Nets: The difficulty of learning long-term dependencies," in *A Field Guide to Dynamical Recurrent Networks*, 2010.

[38] A. Azari, "Bitcoin price prediction: An ARIMA approach," *arXiv*, 2019, [Online]. Available: http://arxiv.org/abs/1904.05315.

[39] D. Michie, "'Memo' functions and machine learning," *Nature*, vol. 218, no. 5136, pp. 19–22, 1968, doi: 10.1038/218019a0.