Journal of COMPUTER AND FORENSIC SCIENCES

CFS 2024, Vol. 3, Issue 1, pp. 22–32
https://doi.org/ 10.5937/jcfs3-48703
Original research paper

An Approach to Optimization of Gated Recurrent Unit with Greedy Algorithm

Patricio Lazcano Muñoz*
SRH University Heidelberg, student
*Corresponding author: patricio.lazmu@gmail.com

Received: January 14, 2024 • Accepted: February 1, 2024 • Published: February 27, 2024.

Abstract: This study focuses on enhancing the performance of Stacked Gated Recurrent Unit (GRU) model in time series data processing, specifically in stock price prediction. The most significant innovation occurs in the integration of a Greedy Algorithm for optimizing hyperparameters such as look-back period, number of epochs, batch size, and units in each GRU layer. Historical stock data from Apple Inc. is utilized for the model's application, emphasizing the model's effectiveness in predicting stock prices. The study methodology involves a sequence of steps, such as data loading, preprocessing, dataset splitting, model construction, and evaluation. The role of the Greedy Algorithm is to iteratively adjust hyperparameters to minimize the Root Mean Squared Error (RMSE) metric, resulting in refining the model's predictive accuracy. The outcomes reveal that the integrated Greedy Algorithm significantly enhances the model's accuracy in predicting stock prices, indicating its potential application in various scenarios requiring precise time series forecasting.

Keywords: stacked gated recurrent unit, greedy algorithm, time series prediction, hyperparameter optimization, stock price forecasting.

1. INTRODUCTION

In the dynamic field of machine learning, Gated Recurrent Unit (GRU) networks have emerged as a reliable and efficient variant of Recurrent Neural Networks (RNNs), known for their capability of handling sequential data [1]. The focus of this paper is on the model variation of Stacked Gated Recurrent Unit, which has been improved with a Greedy Algorithm for an optimal performance. This research delves into the analysis of this specific model, evaluating its effectiveness and efficiency in time series data processing.

The standard RNN architecture, which was a big step forward for handling data sequences, often faces issues like vanishing and exploding gradients, which limit its ability to learn long-range dependencies in data [2]. The introduction of GRU models, a simpler variant of the Long Short-Term Memory (LSTM) networks, has partly addressed these challenges by introducing gating mechanisms. These mechanisms effectively control the flow of

Citation: P L Muñoz, "An Approach to Optimization of Gated Recurrent Unit with Greedy Algorithm," *Journal of Computer and Forensic Sciences*, vol. 3, no. 1, pp. 22–32, 2024, https://doi.org/ 10.5937/jcfs3-48703.

information, making GRUs adept at capturing dependencies in sequential data without excessively complex computations [3].

Stacked GRU models take this a step further by layering multiple GRU units, so the model can learn more complex representations and capture deeper levels of abstraction in the data [4]. This becomes particularly useful for tasks where the patterns in time series data are either not obvious right away or are very complex. The incorporation of a Greedy Algorithm in this model also aims to optimize the selection of hyperparameters, a crucial aspect that significantly influences the performance and accuracy of neural networks. Hyperparameters, such as the look-back period, number of epochs, batch size, and the number of units in the model, greatly influence the learning process and the model's ability to apply what it learns from the training data to new, unseen data.

The primary objective of this paper is to present an in-depth analysis of the Stacked Gated Recurrent Unit model, equipped with a Greedy Algorithm, in the context of time series data. The emphasis will be on the model's architecture, the rationale behind using stacked layers and a greedy approach for hyperparameter optimization, and the performance evaluation in terms Root Mean Squared Error (RMSE) metric as a measure of predictive accuracy.

2. BACKGROUND AND RELATED WORK

Before getting into the depths of Stacked Gated Recurrent Unit networks, it is essential to establish a solid understanding of Recurrent Neural Networks and the challenges they aim to address. RNNs are a type of artificial neural network designed to process sequential data, making them ideal for tasks where the order of information matters, such as in natural language processing, speech recognition, and time series analysis [5].

While RNNs are very promising at identifying patterns in sequences, they struggle with a fundamental limitation known as the vanishing and exploding gradient problems. The gradient is a mathematical concept used to adjust the weights in a neural network during the training process that provides information about how much these weights should be adjusted to enhance the network's performance. These problems appear while RNNs are being trained because older data no longer has the same influence on the network's weight updates [6]. For example, for a vanishing gradient, in essence, as RNNs attempt to learn long-range dependencies in data, the gradients that guide weight adjustments become exceedingly small as they move backward in time across the network, and for an exploding gradient, the other way around, with weight adjustments becoming excessively large.

This phenomenon has significant downsides. When gradients get smaller, the network struggles to propagate error signals effectively during training. As a result, RNNs tend to focus disproportionately on recent information, frequently ignoring or exaggerating the importance of prior data elements. In the context of sequential data analysis, this limitation is particularly problematic, as it prevents the model's ability to capture complex patterns and dependencies that cover wide temporal ranges [7].

To address the vanishing and exploding gradient problem, the GRU emerged as an innovative solution, being a specialized variant of RNNs that address the challenges posed by long-range dependencies and stabilize the training process by managing gradient flow more effectively.

The GRU architecture has gating mechanisms, each designed to regulate the flow of information through the network. These gates are the update gate and the reset gate. The update gate determines how much of the past information should be retained, while the reset gate decides the extent to which new input should be incorporated. This dual-gate mechanism enables GRUs to capture extended temporal dependencies efficiently, offering precise control over what information is retained and forgotten [8].

3. DATASET AND METHODS

The study starts with the acquisition of historical stock data, which in this case is from Apple Inc. (AAPL), within the date range of January 1, 2010 to September 9, 2023. This process focused on obtaining 'Close' prices that reflect the stock's daily closing value. Then, the collected data goes through a comprehensive preparation stage. Here, normalization is a key process implemented using Min-Max scaling, a technique pivotal in ensuring data uniformity and consistency [9]. Such preprocessing is foundational, setting the stage for the model to effectively learn and make accurate predictions.

After that, partitioning the dataset into discrete training and test sets is required for efficient model evaluation. Therefore, the dataset is divided, containing 80% of it for the training set and the remaining 20% for the test set. By using this data separation technique, the model will be trained on historical data while being able to evaluate the accuracy of the prediction with the actual data.

Then it proceeds to the model construction phase. Employing TensorFlow's Keras API, a Stacked GRU model is architected, consisting of multiple layers of GRU units. This design plays a key role in capturing complex patterns in the sequential data, an essential characteristic for the complex task of stock price prediction [10]. Each layer in this stacked configuration is meticulously configured to contribute to the overall learning capability of the model.

Crucial to the model's development is the Greedy Algorithm, used for hyperparameter optimization. This algorithm plays an important role in iteratively refining the model's parameters, including the look-back period, number of epochs, batch size, and units in each GRU layer. The objective of it is to iteratively explore various combinations of these parameters so the model can be trained on the pre-processed dataset to make predictions and fine-tune its internal parameters in the process, aiming to identify the combination that minimizes the RMSE value on the test dataset.

Finally, the visualization and analysis of the model's predictions are added to translate the model's numerical predictions into a more interpretable format. By comparing the model's predicted stock prices against actual prices, a clearer picture of the model's performance is obtained, providing valuable insights into its practical efficacy.

So basically, the methodology and implementation used can be explained with the following steps [11]:

1) Data Loading: The initial step involves the acquisition of historical stock market data for AAPL. This data, sourced using the yfinance library, provided a comprehensive view of the stock's performance over a significant period.

2) Data Preprocessing: The next step, after the data retrieval, begins by extracting the Close prices, representing the closing prices of the company's stock for each trading day, along with the corresponding dates. With these values, the data is then normalized. This standardization process ensures uniformity and consistency across the data. Min-Max scaling is applied, transforming the data into a common scale.

3) Dataset Splitting: Following the preprocessing, the dataset is divided into two distinct sets: a training set and a test set. This split data is balanced, with around 80% of it being the training set, while the remaining 20% forms the test set.

4) Model Construction: After it, the flow advances to model construction, creating a Stacked GRU model that features multiple GRU layers, each tailored to capture and learn from different levels of data abstraction. This stacking of GRU units is critical to enhancing the model's ability to discern and learn from complex data patterns prevalent in stock price movements.

5) Model Training and Evaluation (with the Greedy Algorithm): The Greedy Algorithm's role is central to the optimization of various hyperparameters, including the look-back period, epochs, batch size, and the number of units in the GRU layers. This algorithm employs an iterative approach, testing different combinations of these parameters to find the one that minimized the RMSE value. This step involves feeding the prepared training features and targets into the Stacked GRU model to then evaluate it on the test set, employing RMSE as the key metric to assess the performance of its predictions.

6) Visualization and Analysis: The final phase involved visualizing the model's predictions in comparison with the actual stock prices. This step was crucial in providing a clear and interpretable representation of the model's performance, offering valuable insights into its efficacy and accuracy in predicting stock prices.

Also, we consider that the general workflow of the Greedy Algorithm is:

1) Initialization: We start with the first reset by initializing a set of the following hyperparameters: look back value, number of epochs, batch size, and number of units, which we will refer to as LOOK_BACK_VAL, EPOCHS_VAL, BATCH_SIZE_VAL, and UNITS_VAL, respectively. They are assigned random values within predefined ranges adapted to each hyperparameter.

2) Initial RMSE Calculation: For this initial set of hyperparameters, we calculate the RMSE of the model, which is used as a reference point, and designate the given set of hyperparameters as the currently most optimal set.

3) Mutation and Sub-Set Evaluation: Then we mutate each hyperparameter from the currently most optimal set, one at a time, while keeping the others constant. This mutation picks a new random value within the allowed range for that specific hyperparameter. For each mutated version, we create a new sub-set of hyperparameters and calculate its RMSE, so we will end up with several sub-sets equivalent to the number of hyperparameters.

4) Comparing RMSE and Selecting the Currently Most Optimal Step: After calculating the RMSE values from all the sub-sets, we start comparing the RMSE of each of these sub-sets with the current minimum RMSE of the reset. If a sub-set has a lower RMSE value, then it is selected and becomes the new currently most optimal set of hyperparameters, and its RMSE becomes the new current minimum RMSE. Then, the mutation process (step 3) is repeated for the new set. If none of the sub-sets have a lower RMSE, then we

end this reset with the last currently most optimal set and RMSE and continue with the next reset.

5) Algorithm Cycling and Final Selection: This whole process is repeated for a total of 16 resets (starting with RESET 0 and concluding with RESET 15). Each reset potentially gives us a final most optimal set of hyperparameters with its RMSE. After completing all resets, we select the final most optimal set among these that has the lowest RMSE value across all resets. This set is then adopted as the optimal configuration for the respective RNN model.

The workflow of the Greedy Algorithm is represented in **Figure 1**.

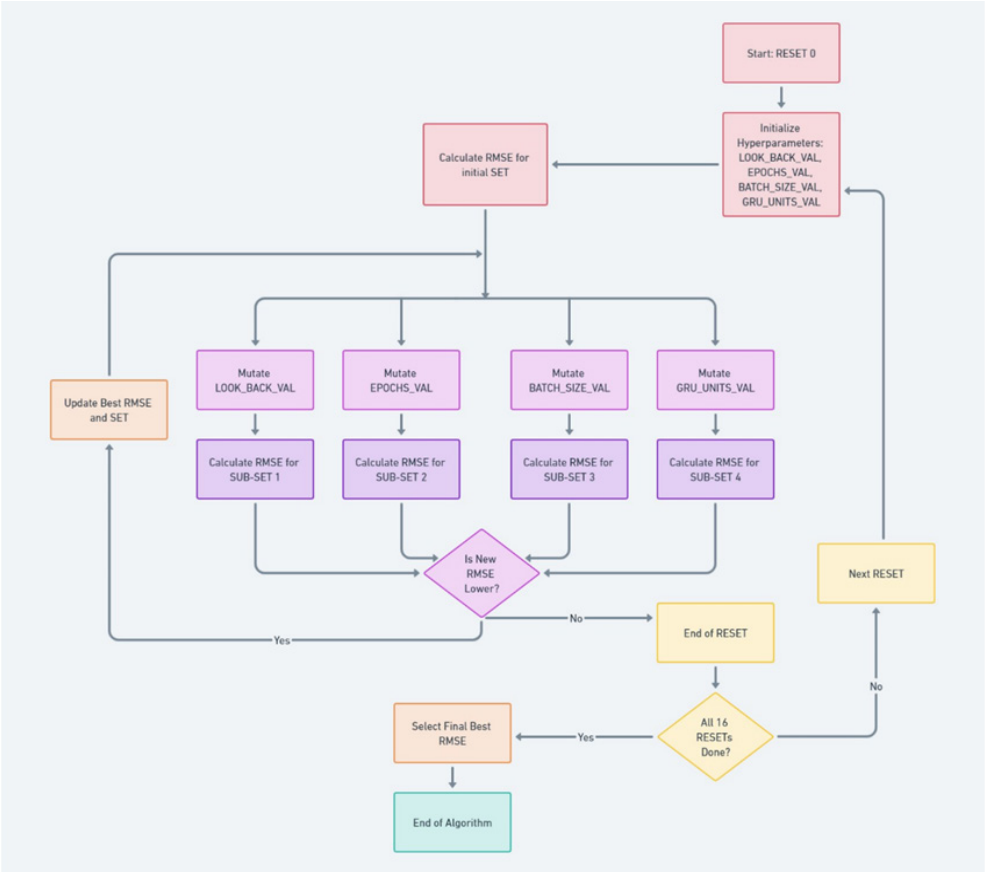


Figure 1. *Greedy Algorithm workflow.*

4. RESULTS

The results that were obtained from experimenting with the prediction of stock prices for **Apple Inc.**, by training the model with the historical data from "**01-01-2010**" to "**17-12-2020**" (80% from total) and using the dates from "**18-12-2020**" to "**08-09-2023**" (20% from total) as testing data. The objective was to identify the impact of trying different combinations of hyperparameter, mutating their values, evaluating its performance with the RMSE metric and making **resets** or **restarts** to be able to check different starting points. The hyperparameters being examined were [12]:

- **LOOK_BACK_VAL**: The number of previous time steps used for prediction. With a random value assigned from 1 to 100.
- **EPOCHS_VAL**: The number of training iterations. With a random value assigned from 1 to 200.
- **BATCH_SIZE_VAL**: The number of training examples used in each iteration. With a random value assigned from 2 to 128 (that is divisible by 2).

GRU_UNITS_VAL: The number of GRU units (neurons) in the model. With a random value assigned from 1 to 100.

The hyperparameter values were randomly changed within their respective ranges during each mutation of the Greedy Algorithm, and then the final set of combinations that performed better on each reset was chosen.

The full output values obtained can be observed in **Table 1**.

Table 1. *Output results from Stacked Gated Recurrent Unit model.*

		Stacked GRU						
		Look_back	Epochs	Batch_size	RNN Units	RMSE	Total No. of SETS in RESET	Total Execution Time
RESET 0	Initial SET	85	28	74	28	9,78298979391300	17	35 minutes and 37 seconds
	Final SET	18	127	74	62	3,46307678189240		
RESET 1	Initial SET	7	69	112	14	8,00179284914421	13	
	Final SET	7	69	112	84	2,99117282259637		
RESET 2	Initial SET	71	101	41	50	5,95858386128478	13	
	Final SET	19	101	42	50	3,47578559209489		
RESET 3	Initial SET	44	80	60	78	11,11752756590960	9	
	Final SET	11	86	60	78	3,63990671872683		
RESET 4	Initial SET	75	43	86	24	10,78929398838540	9	
	Final SET	75	43	86	73	4,48976711017981		
RESET 5	Initial SET	48	138	62	54	7,41746525515739	13	
	Final SET	2	162	62	54	2,83994332212022		
RESET 6	Initial SET	10	158	70	43	5,12979017014056	9	
	Final SET	10	158	70	28	3,58318293961015		
RESET 7	Initial SET	34	21	88	40	6,01884659174220	17	
	Final SET	34	133	88	40	3,92937844057308		
RESET 8	Initial SET	45	37	128	88	6,42584658538581	17	
	Final SET	45	37	56	71	3,84006890241281		
RESET 9	Initial SET	72	32	94	79	8,63722306087351	13	
	Final SET	72	197	122	79	3,85035521106658		
RESET 10	Initial SET	35	141	30	60	4,70368069100479	9	
	Final SET	16	141	30	60	3,48418875613201		
RESET 11	Initial SET	65	154	80	8	10,81405337164620	17	
	Final SET	6	154	80	89	2,94384831986923		
RESET 12	Initial SET	54	68	26	41	10,30640565807570	9	
	Final SET	28	68	26	41	3,82460735936765		
RESET 13	Initial SET	5	137	22	70	4,94286017178786	13	
	Final SET	5	129	104	70	2,87246724327228		
RESET 14	Initial SET	14	38	124	18	9,46690534979822	13	
	Final SET	14	117	124	53	3,71814035974593		
RESET 15	Initial SET	1	19	106	69	3,47084584026259	13	
	Final SET	**1**	**183**	**106**	**71**	**2,75106347799420**		

In this experimentation, the best combination of hyperparameters happens in **RESET 15**, with a RMSE value of **2,7510634779942**, Having the final set combination:

- **LOOK_BACK_VAL** = 1, **EPOCHS_VAL** = 183, **BATCH_SIZE_VAL** = 106, **GRU_UNITS_VAL** = 71.

That leads to the plot that compares the Actual Prices from the historical data with the Predicted Prices, as depicted in **Figure 2**.

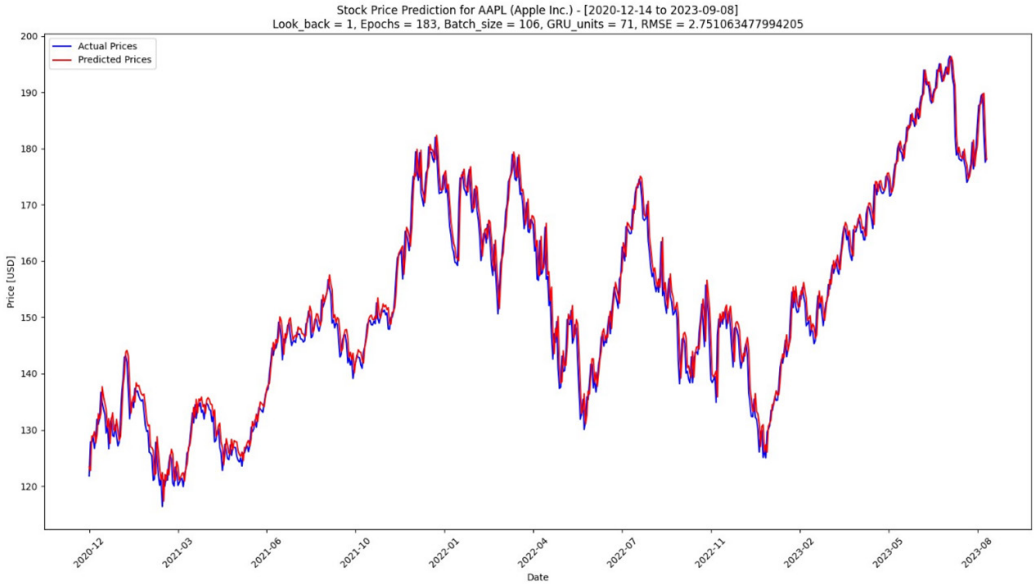
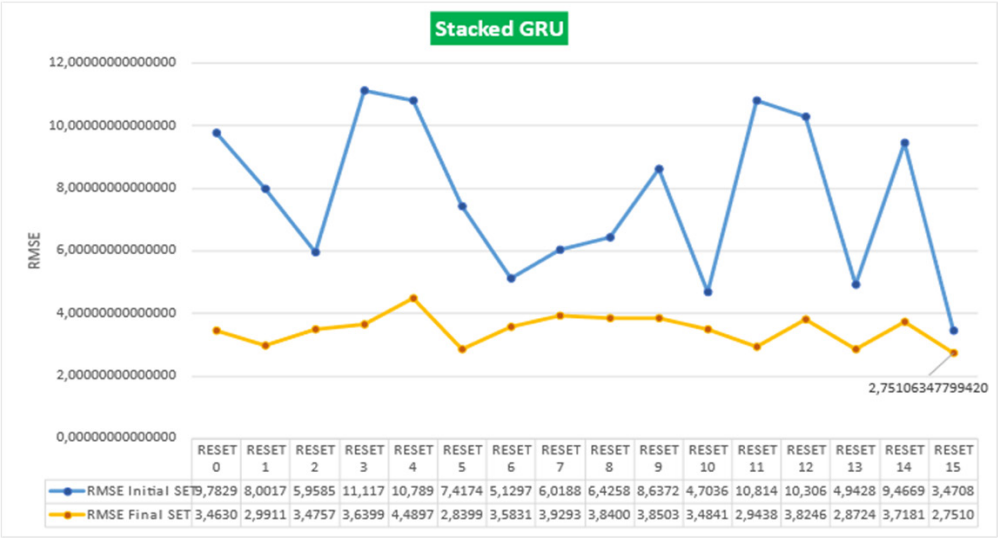


Figure 2. *Stock Price Plot from the most optimal set of Stacked GRU model.*

We can observe that the difference between both lines is small, approaching the predictions close to the real value.

A more visually detailed behavior depicted in Table 1 can be seen in the graph comparison of RMSE values of the initial set and final set for each reset, represented in **Figure 3**.



	RESET 0	RESET 1	RESET 2	RESET 3	RESET 4	RESET 5	RESET 6	RESET 7	RESET 8	RESET 9	RESET 10	RESET 11	RESET 12	RESET 13	RESET 14	RESET 15
RMSE Initial SET	9,7829	8,0017	5,9585	11,117	10,789	7,4174	5,1297	6,0188	6,4258	8,6372	4,7036	10,814	10,306	4,9428	9,4669	3,4708
RMSE Final SET	3,4630	2,9911	3,4757	3,6399	4,4897	2,8399	3,5831	3,9293	3,8400	3,8503	3,4841	2,9438	3,8246	2,8724	3,7181	2,7510

Figure 3. *Initial vs Final RMSE of each reset in Stacked GRU model.*

In more detail, we obtained the **Average** and **Standard Deviation** values from the RMSE results of all the initial and final sets, represented in **Table 2**.

Table 2. *Average and Standard Deviation of RMSE from Stacked GRU Model.*

	from Initial SETS	from Final SETS
RMSE Average	7,68650692528199	3,48105958485340
RMSE Standard Deviation	2,513627359	0,485585321

We can observe that the RMSE values from all the final sets are close to each other, so in this case, the Standard Deviation is telling us that the final RMSE values of each reset tend to approach the Average value in this model. Suggesting a stable and reliable model performance across different sets or scenarios.

Also, we obtained the Average and Standard Deviation values from the hyperparameters of the final sets of each reset to analyze their behaviors, as shown in **Table 3**.

Table 3. *Average and Standard Deviation of Hyperparameters from Stacked GRU Model.*

	from Initial SETS	from Final SETS
Look_Back Average	41,5625	22,6875
Look_Back Standard Deviation	27,80160367	23,11051348
Epochs Average	79	119,0625
Epochs Standard Deviation	51,49498357	48,05339912
Batch_size Average	75,1875	77,625
Batch_size Standard Deviation	33,69118332	30,85638778
GRU Units Average	47,75	62,6875
GRU Units Standard Deviation	24,95195383	17,26545202

Here we can notice the following details for each hyperparameter:

- **Look_Back**: The reduction of the average and standard deviation implies that the algorithm often favored lower Look_Back values, finding them more effective for minimizing RMSE.
- **Epochs**: The increase in average, alongside a still notable standard deviation in the final sets, suggests that higher Epoch values are generally more favorable, but with significant variability across different resets.
- **Batch_size**: The minor change suggests that Batch_Size does not significantly impact the model's performance, as seen by comparatively stable values after optimization.
- **GRU Units**: The increase in average and decrease in standard deviation suggest that higher GRU unit values tend to result in better performance, with the algorithm converging more consistently on these higher values.

Here, we need to keep in mind that the best combination of hyperparameters may not be the one that individually performs the best because there could be a trade-off between different hyperparameters. For example, a larger value in **look_back** might require a larger value of **GRU units** to capture more complex patterns. Overall, experimenting with various combinations in this research has given us a clearer and better understanding of how the model works.

5. CONCLUSIONS

The research of the Stacked Gated Recurrent Unit model integrated with a Greedy Algorithm demonstrated a significant advancement in time series data analysis, particularly in stock price prediction, providing a valuable understanding of the dynamics of hyperparameter optimization and model performance.

The relevant contribution of this research resides in the clever use of the Greedy Algorithm, by mutating the hyperparameters until a point where further reduction of the Root Mean Squared Error value is no longer achievable, facilitating in this way an efficient navigation of the hyperparameters range, leading to a notable increase in the accuracy of predictions.

The introduction of resets is also a key aspect of the methodology, allowing the existence of multiple starting points and providing the opportunity to explore a diverse range of hyperparameter combinations. The selection of the minimum RMSE value out of 16 resets represents an effective method for identifying the most effective model configuration.

The comparison of Actual Prices from historical data with Predicted Prices shows the great predictive accuracy of the model, as demonstrated by the small variations between these values. This suggests that the Stacked GRU model, when optimally configured, can closely simulate data from the real world, which is a significant achievement in the field of time series prediction.

The most effective combination of hyperparameters was determined in RESET 15, with the RMSE value reaching 2,7510634779942. Including a LOOK_BACK_VAL of 1, EPOCHS_VAL of 183, BATCH_SIZE_VAL of 106, and GRU_UNITS_VAL of 71. This indicates a preference for a specific configuration that maximizes predictive performance while minimizing error.

The research revealed that Batch Size changes had a minor impact on model performance, suggesting a certain amount of resistance to this parameter. On the other hand, a better performance was frequently associated with greater GRU unit values, indicating their critical role in the model's predictive capabilities.

The RMSE values from all Final sets were closely grouped in this model, with a low standard deviation, reflecting a stable and reliable model performance across various scenarios. This consistency is crucial for practical applications, as it implies that the behavior of the model is reliable and predictable.

A noticeable pattern was the algorithm's tendency for lower Look_Back values, which were more effective in minimizing RMSE. In contrast, higher Epoch values were generally

favorable but with significant variability, indicating a complex connection between epochs and model accuracy.

In essence, this research has not only deepened our understanding of Stacked GRU models but has also made a vital contribution to the field of machine learning. The innovative application of the Greedy Algorithm for hyperparameter optimization and the methodical approach of using multiple resets have proven to be crucial in achieving remarkable predictive accuracy with the Stacked GRU model. This work provides the guidance for utilizing these kinds of models in a variety of situations where high precision in time series forecasting is required.

FUNDING

This research received no external funding.

INSTITUTIONAL REVIEW BOARD STATEMENT:

Not applicable.

INFORMED CONSENT STATEMENT:

Not applicable.

CONFLICTS OF INTEREST:

The author declares no conflict of interest.

REFERENCES

[1] Z. Johnson, "Exploring Gated Recurrent Unit: A Powerful Neural Network Approach", PicDataset, July 23, 2023. [Online]. Available: https://picdataset.com/2021/exploring-gated-recurrent-unit-a-powerful-neural-network-approach/ [Accessed January 10, 2024].

[2] S. Zivkovic, "RNN_Tackling Vanishing Gradients with GRU and LSTM", DataHacker, September 24, 2020. [Online]. Available: https://datahacker.rs/005-rnn-tackling-vanishing-gradients-with-gru-and-lstm/ [Accessed January 10, 2024].

[3] Data Science Team, "Gated Recurrent Units – Understanding the Fundamentals", DataScience, June 02, 2022. [Online]. Available: https://datascience.eu/machine-learning/gated-recurrent-units-understanding-the-fundamentals/ [Accessed January 10, 2024].

[4] F. Pan, J. Li, B. Tan, C. Zeng, X. Jiang, L. Liu, and J. Yang, "Stacked-GRU Based Power System Transient Stability Assessment Method", MDPI, August 09, 2018. [Online]. Available: https://www.mdpi.com/1999-4893/11/8/121/htm [Accessed January 10, 2024].

[5] V. Rai, "Recurrent Neural Networks (RNNs): Understanding Sequential Data Processing", InterviewKickstart, December 27, 2023. [Online]. Available: https://www.interviewkickstart.com/learn/recurrent-neural-networks-sequential-data-processing [Accessed January 10, 2024].

[6] Baeldung, "Prevent the Vanishing Gradient", Baeldung, January 11, 2024. [Online]. Available: https://www.baeldung.com/cs/lstm-vanishing-gradient-prevention [Accessed January 10, 2024].

[7] Aishwarya, "Introduction to Recurrent Neural Network", GeeksforGeeks, December 04, 2023. [Online]. Available: https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/ [Accessed January 10, 2024].

[8] S. Kostadinov, "Understanding GRU Networks", Towards Data Science, December 16, 2017. [Online]. Available: https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be [Accessed January 10, 2024].

[9] S. Selvidge, "Min-Max Data Normalization in Python: Best Practices", CeleryQ, September 20, 2023. [Online]. Available: https://celeryq.org/min-max-normalization-python/ [Accessed January 10, 2024].

[10] T. B. Shahi, A. Shrestha, A. Neupane, and W. Guo, "Stock Price Forecasting with Deep Learning: A Comparative Study", MDPI, August 27, 2020. [Online]. Available: https://www.mdpi.com/2227-7390/8/9/1441 [Accessed January 10, 2024].

[11] Y. Tang, "Build a GRU RNN in Keras", PythonAlgos, January 02, 2022. [Online]. Available: https://pythonalgos.com/2022/01/02/build-a-gru-rnn-in-keras/ [Accessed January 10, 2024].

[12] J. Brownlee, "Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras", Machine Learning Mastery, August 07, 2022. [Online]. Available: https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/ [Accessed January 10, 2024].