IJHSR

ijhighschoolresearch.org

■ REVIEW ARTICLE

Analyzing Efficiency-based Hyperparameter Tuning Optimization Methods on LSTMs for Generative ARIMA Models

Anon Taulananda

Singapore International School of Bangkok, 498 11 Rotchanamin Alley, Wang Thonglang, Bangkok, 10310, Thailand; tytaulananda@gmail.com

ABSTRACT: In the deep learning field, Long Short-Term Memory networks (LSTMs) are a type of recurrent neural network (RNN) that use gates to control and retain important information, enabling them to handle long-range dependencies and effectively forecast time series models. Despite their effectiveness, LSTMs can be further optimized for increased accuracy and efficiency. Optimizing these networks involves fine-tuning hyperparameters, which are external parameters determined manually. While this can be done through trial and error, it is more efficiently accomplished using hyperparameter optimization algorithms such as random search, Bayesian optimization, and Hyperband. This paper compares these three algorithms, finding that random search achieved low mean-squared error but raised concerns about reproducibility and consistency, Hyperband excelled in resource efficiency but struggled with optimal configuration, and the Bayesian algorithm offered consistency and a smooth learning process at a higher computational cost and required parameter adjustments.

KEYWORDS: Robotic and intelligent machines; Machine learning; Neural network; LSTM; Hyperparameter optimization.

■ Introduction

Neural networks are a subset of artificial intelligence that aims to mimic the human brain. It has the capability to learn and adapt allowing it to classify, predict, process, and generate a variety of data. This feature is the reason behind why they are applied to a variety of prominent fields, some notable examples are medical diagnosis,[2] stock prediction,[3] chat bots,[4] and autonomous robotics.[5] Long Short-Term Memory networks (LSTMs) are a type of recurrent neural network (RNN) introduced in 1997 with the aim of tackling sequence prediction problems.[6] They use a system of gates to maintain and update information over long sequences, allowing them to capture long-term dependencies effectively. This makes LSTMs excel at handling time series datasets, such as those used in autoregressive integrated moving average (ARIMA) models.

Creating and configuring a LSTM may be time-consuming and a tedious part of that process is choosing optimal parameters, especially hyperparameters. Hyperparameters are external parameters manually set by the user, such as number of hidden LSTM layers, number of hidden units, learning rate, and type of activation function for the dense layers just to name a few. Fine-tuned hyperparameters lead to a high-performing neural network, whereas, nonoptimal hyperparameters may lead to one that is lousy and inefficient, especially in fields where reliability and accuracy of prediction is crucial.[7] This paper aims to enhance LSTMs across various applications by tuning hyperparameters using different optimization algorithms and comparing the results, providing insights that could lead to more accurate and resource-efficient time series forecasting models while reducing the time required to create and configure an LSTM.

Hyperparameter optimization can be done manually through exhaustive means, but novel and emerging optimization techniques are being applied to obtain these optimal hyperparameters,[8] This research paper tests three of these optimization techniques: Bayesian optimization, Hyperband, and random search, with the aim to compare the results and the resources required. Based on prior research, the Bayesian optimization is anticipated to yield the most optimal hyperparameters utilizing its probabilistic approach. Hyperband is expected to excel in efficiency due to its bandit-based strategy, while random search, serving as a baseline, is predicted to produce suboptimal results due to its lack of a systematic exploration strategy.

The main results of this research indicate significant differences in the performance and efficiency of the tested optimization algorithms. Random search minimized the mean-squared error to 1.56, performing 62.0% better than the baseline, but raised concerns about reliability and reproducibility. Hyperband demonstrated excellent resource efficiency with a mean-squared error of 2.12, 48.4% better than the baseline, and faster runtimes. Bayesian optimization achieved a mean-squared error of 1.68, 59.2% better than the baseline, with the longest runtime but showed great consistency and smooth learning curves. These findings highlight the strengths and weaknesses of each optimization technique in improving LSTM model performance.

These algorithms are all computationally expensive and a scope is required for both the number of hyperparameters tested and the maximum number of runs these hyperparameter optimization techniques can perform. Naturally, these are dependent on the complexity and nature of the dataset, LSTMs running on different datasets will have different sets of optimal hyperparameters. To reduce dependencies on dataset and to

DOI: 10.36838/v6i9.2

focus purely on the performance of each optimization technique this research paper will be utilizing a generated ARIMA dataset.

Methods

Neural network:

The chosen deep learning neural network to train the dataset is the Long-short term memory (LSTM).[6] It is a type of recurrent neural network (RNN) that aims to solve the vanishing (or exploding) gradient problem and is proficient in dealing with time-series datasets due to its capability of remembering long-term dependencies. Other than time-series datasets, it's also utilized for language processing, speech recognition, autonomous robots and more.

An LSTM unit is composed of three key gates,[6] each playing a crucial role in information processing. The forget gate evaluates whether to retain or discard information from the previous step. The input gate assesses the significance of the current input, influencing the extent to which it is integrated into the cell state. Lastly, the output gate determines the subsequent hidden state by considering both the updated cell state and the current input. LSTMs consist of numerous LSTM units, and their architecture allows the neural networks to excel in a wide array of tasks, particularly those demanding the handling of intricate long-term dependencies.

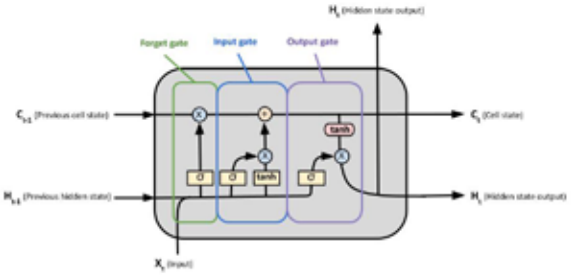

Figure 1: LSTM Unit.

Optimization algorithm:

All 3 optimization algorithms were imported from the keras tuner library,[9] dataset generation and predictions were graphed using numpy and the mean-squared error per epoch graphs were graphed using Tensorflow's tensorboard.

Random search:

Random search uses a stochastic approach to search for optimal hyperparameters.[10] The algorithms randomly sample sets of hyperparameters, compare them and return the best performing set. Each hyperparameter has an equal chance of being selected allowing it to occasionally stumble across near optimal hyperparameters. Random search is a popularized alternative to the exhaustive grid search due to its more diverse exploration and time efficient search, while also often producing better results. A drawback to the random search is its low reliability and replicability. Results will vary depending on each run, therefore, it isn't an optimal algorithm for cases where consistency and accuracy is required.

Hyperband:

Hyperband uses a bandit-based approach to search for optimal hyperparameters.[11] The algorithm combines the random search with successive halving, a resource-allocation strategy that allocates resources to only promising configurations. This further boosts efficiency and control costs while obtaining the same or more optimal hyperparameters than the random search. The algorithm allows the user to set the maximum amount of resource used per configuration which proves useful when optimizing hyperparameters with limited resources. However, with the goal of efficiency, the Hyperband algorithm compromises on its exploration for an optimal set of hyperparameters.

Bayesian optimization:

Bayesian optimization uses a probabilistic approach to search for optimal hyperparameters.[12] The algorithm contains a probabilistic surrogate model and an acquisition function. This paper will be using the Gaussian process as the surrogate model which is known for its flexibility and the expected improvement (EI) acquisition function which excels at balancing exploration and exploitation. Bayesian optimization begins with a small number of initial random samples, allowing it to construct a probabilistic model of the objective function. The algorithm then iteratively refines its model, selecting new sets of hyperparameters that are likely to achieve better results than the current best-known value of the objective function. Due to the probabilistic nature, results achieved by the Bayesian optimization may be inconsistent. Additionally, users may be required to tune the optimization model to balance the exploration and exploitation to ensure performance.

Dataset:

This paper uses a generative auto-regressive integrated moving average (ARIMA) model to create a dataset of 1,000 data points. The ARIMA model works by using all past time-series data to generate or predict the next data point and repeating this process until all data points are generated. Each ARIMA data point is derived by first generating a sequence of returns from an ARMA model with AR coefficients [1, -0.4], MA coefficients [1, -0.8] and a random seed of 42, then calculating the cumulative sum of those returns. A constant drift term is also added to each data point to simulate a realistic financial dataset.[20] The dataset is then split into a training set and a validation set, 800 data points are used for training (80%), and 200 data points are used for validation (20%). The performance of the tuned LSTM will be tested on the validation set with the aim to achieve the lowest mean squared error with the least average time per execution.

Evaluation metrics:

1. Mean squared error:

$$MSE = \frac{1}{n}\sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2$$

The mean squared error is the average squared difference between the predicted value and the actual value. The set aim for each of the hyperparameter optimization algorithms is the minimization of the validation mean squared error, therefore, performance will be evaluated on the mean squared error when running the tuned LSTM through the prediction dataset.

2. Average training time per trial:

The efficiency of the optimization algorithm will be evaluated by comparing the average program run time per trial.

DOI: 10.36838/v6i9.2

The total computation time includes the duration for both algorithm preparation and the time spent on TensorBoard callbacks. This comprehensive evaluation provides insights into the overall efficiency of the optimization process.

3. Consistency:

Consistency and reproducibility of results across runs is a crucial aspect to an optimization algorithm. This metric provides insight on reliability that proves important for real-life application and also aims to visualize how the algorithm adapts and learns from previous trials by seeing the fluctuations in mean-squared errors of each trial and the smoothness of the learning process.

4. Hyperparameters selection:

The optimal hyperparameters generated by the optimization algorithm should not only enhance performance but also demand minimal resources. Therefore, this paper will additionally assess the runtime of each LSTM model tuned with the hyperparameters derived from the optimization algorithms.

5. Convergence speed:

How fast each model reaches the lowest possible mean-squared error for each run and how many epochs it takes to reach it. The average mean-squared error of the initial epochs will also be analyzed to find insights on model stability and early performance indicators.

6. Generalization:

Overfitting is a common issue in modeling. A high-performing model should not only perform well on the training dataset but also make accurate predictions on unseen data. The ability of each model in reducing the problem of overfitting and how generalized the models are will be analyzed to gain insight on suitability for real-life applications.

Hyperparameters search space:

The LSTM model is initially built with an input layer, a hidden LSTM layer, and a final dense layer. The model has a loss function and an evaluation metric of mean squared error and the optimizer 'adam'. These are kept constant while optimizing the following hyperparameters:

1. Number of input units:

Input units are the neurons located in the input layer that directly receives and processes the input data while capturing and representing complex patterns in input data. Varying complexity of data requires a different number of input units, therefore, it has to be optimized to adapt to input complexity and prevent both overfitting and underfitting.

2. Number of hidden LSTM layers and units:

Hidden layers are located between the input and the output layers, and each layer has hidden LSTM units that extracts relevant features while learning patterns in the dataset. Optimizing the number of layers and units is done to improve the model's capabilities to capture patterns while still generalizing well.

3. Dropout rate:

Dropout rate dictates the percentage of connections or neurons randomly dropped by the neural network. Optimizing the dropout rate is crucial for mitigating overfitting issues, which is when the model learns training data too well, capturing noise and fluctuations instead of the underlying pattern leading to poor performance when running the model through unseen data.

4. Dense layer's activation function:

Activation functions are functions that introduce non-linearity to the model, restrict output and allow models to learn more complex and high-dimensional data. Each activation function is suitable for different tasks and optimizing them will result in faster convergence and reduction in issues like exploding gradients. Here are the following non-linear activation function used for this paper:

a) Rectified linear units (ReLU):

ReLU solves the exploding gradient problem and is a popular function for all use cases; it is a simple non-linear activation function with the formula:

$f(x) = max(0,x)$.

It returns the input value unless it is negative, in which case it returns 0. Therefore its range is from 0 to infinity.

b) Sigmoid:

Sigmoid is often used to predict probability; it is also a popular non-linear activation function with the formula:

$$f(x) = \frac{1}{1+e^{-x}}$$

It has a smooth "S" shape and ranges between 0 and 1, making it suitable for dealing with probabilities.

c) Hyperbolic tangent function (tanh):

Tanh is often used for classification; it is a more complex non-linear activation function with the formula:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

In contrast to the sigmoid function, this function is centered around the origin and can produce negative values. Its overall shape resembles that of the sigmoid, yet with a steeper slope, and it ranges from -1 to 1.

$$\text{Hyperparameters} = \begin{cases} \text{LSTM layers} & \text{1-3 (Step: 1)} \\ \text{LSTM units} & \text{32-512 (Step: 32)} \\ \text{Input units} & \text{32-512 (Step: 32)} \\ \text{Dropout rate} & \text{0.0-0.5 (Step: 0.1)} \\ \text{Activation function} & \text{ReLU, tanh, Sigmoid} \end{cases}$$

Total number of possible hyperparameter configurations: $3 \times 16^3 \times 16 \times 6 \times 3 = 3{,}538{,}944$. Due to the large search space, the paper refrained from employing exhaustive search methods like grid search or more sophisticated algorithms such as evolutionary algorithms, which could pose challenges in terms of computational constraints and time limitations. Therefore, the optimization algorithms with aims of efficiency are used for this research paper instead.

Results and discussion

These optimization methods are all Keras tuners, imported from the keras library. They are run on Google Collaboratory with the T4 GPU engine. They were all given 100 trials with a maximum of 30 epochs per trial and a batch size of 128 with the objective of minimizing the validation mean-squared error. Serving as a baseline, the untuned model achieved a mean-squared error of 4.11 on the predictive dataset.

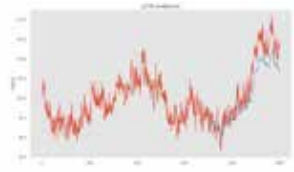

Figure 2.1: Untuned model.

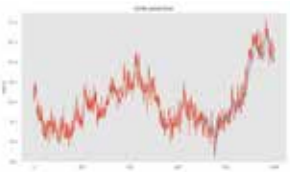

Figure 2.2: Random Search.

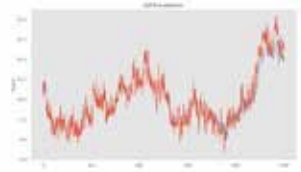

Figure 2.3: Hyperband.

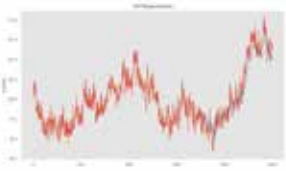

Figure 2.4: Bayesian optimization.

Random Search:

The random search randomly selects and tests hyperparameter configurations from the search space. On its best run it was able to minimize the mean-squared error to an impressive 1.56 when running on the predictive dataset. It had a total runtime of 31 minutes and 31 seconds and an average time of 18.9 seconds per trial. The best hyperparameter configuration found utilized a total of 960 units (LSTM and input units). Although producing the lowest mean-squared error and performing 62.0% better than the baseline, trials were inconsistent and with no search system it struggles with reliability and reproducibility compared to other optimization methods seen from figure 3.2. The graph depicts an erratic and less smooth learning process performed by the model on the validation dataset with observable fluctuations and anomalies. While there is a general convergence around the 12th epoch, the initial epochs on the validation dataset exhibit noticeably mean-squared error and greater variation compared to alternative optimization algorithms. Comparing figure 3.1 and figure 3.2 we can deduce that with a dropout rate of 0.2 the model generalized relatively well while keeping overfitting to a minimum shown by the similar convergence value, graph shape and the absence of large gaps between the two graphs with a few outliers.

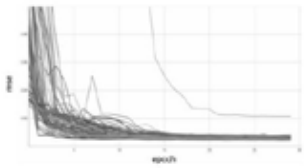

Figure 3.1: Training Epoch vs MSE (RS).

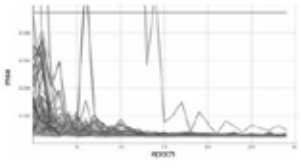

Figure 3.2: Validation Epoch vs MSE (RS).

Hyperband:

Hyperband samples various hyperparameter configurations and allocates resources to promising configurations while discarding less favorable ones. For the paper the cut-out factor was set to 2 and a maximum resource allocation of 30 epochs per trial. On its best run it was able to minimize the mean-squared error to 2.12 (48.4% better than baseline) when running on the predictive dataset. It had a total runtime of 18 minutes and 25 seconds and an impressive average time of 12.0 seconds per trial. The best hyperparameter configuration found utilized a total of 928 units (LSTM and input units). This tradeoff between performance and efficiency was expected, however, the algorithm still obtained a great balance between exploration and exploitation and was able to return promising results in significantly less time. The algorithm's bandit-based approach was computationally efficient, considerably reducing the number of epochs per trial; Throughout the runs only 15 sets of promising configurations (16.3%) made it to the maximum of 30 epochs whereas a staggering 42 less promising configurations (45.7%) were discarded before the 5th epoch. According to figure 4.2, the model showed a smooth learning process and a relatively fast convergence at around 9 epochs. Moreover, the mean squared error of the initial epochs on the validation dataset were surprisingly low and there were minimal fluctuations and outliers throughout the trials indicating a stable algorithm. Regarding generalization, by comparing the graph shape, the convergence value and overall structure of figure 4.1 and 4.2 hyperband performed relatively well on unseen data and showed little to no signs of overfitting.

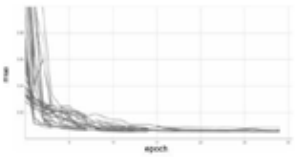

Figure 4.1: Training Epoch vs MSE (HB).

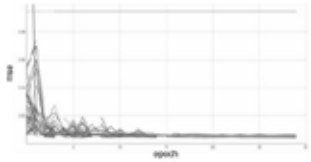

Figure 4.2: Validation Epoch vs MSE (HB).

Bayesian Optimization:

Bayesian optimization uses random samples of hyperparameter configurations to create a probabilistic model for the search space. Due to the large search space we allowed the model to be more explorative by setting the beta parameter to a 7 and number of initial sample points to 10. On its best run it was able to minimize the mean-squared error to 1.68 (59.2% better than baseline) when running on the predictive dataset. It had a total runtime of 34 minutes and 12 seconds and an average time of 20.5 seconds per trial. Although the least time efficient, the bayesian optimization performed relatively well and showed great consistency in results. The best hyperparameter configuration found utilized a total of 1056 units (LSTM and input units). Despite one outlier, the Bayesian algorithm converged relatively rapidly at around 7 epochs and showed a very smooth learning process with the lowest initial mean-squared error. It was also the least erratic, showing the smoothest and steepest graph shape with the fewest fluctuations. With its probabilistic approach, it was able to generate a relatively representative model of the search space and by referring to the model it was able to pick out hyperparameter configurations that had potential and produced similar optimal results. Additionally, comparing Figure 5.1 and Figure 5.2, it showed no overfitting with a nearly symmetrical learning curve and the same convergence value displaying its capabilities on performing on unseen data.

DOI: 10.36838/v6i9.2

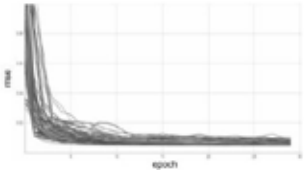

Figure 5.1: Training Epoch vs MSE (BO).

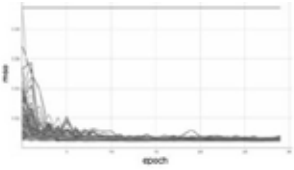

Figure 5.2: Validation Epoch vs MSE (BO).

Table 1: Summary of Performance.

	Random Search	Hyperband	Bayesian optimization
Input units	224	448	512
First layer units	512	448	512
Number of additional layers	1	1	1
Second layer units (Added)	224	32	32
Dropout rate	0.2	0.4	0.4
Dense layer's activation function	ReLU	ReLU	ReLU
Mean-squared error	1.56	2.12	1.68
Average time per trial	18.9	12.0	20.5
Iterations	100	92	100
Total epochs	3000	970	3000

The results show that the three optimization algorithms were able to find similar hyperparameter configurations that significantly improved the LSTM model. They all agreed that 1 additional layer of LSTM and ReLU as the activation function for the dense layer was optimal. The initial hypothesis was partially correct as the results highlighted Hyperband's remarkable efficiency and the Bayesian algorithm's unmatched consistency. However, the random search with its non-methodological approach was able to sample the hyperparameter configuration that returned the lowest mean-squared error. Although all three hyperparameter optimization algorithms are viable and generalized well, the results provided meaningful insights and a guideline for other researchers or users with limited computational resources when selecting between these 3 efficiency-based algorithms based on their weight factors and tradeoffs.

Conclusion

This research paper aimed to provide an objective and thorough analysis of three hyperparameter optimization algorithms—Random search, Hyperband, and Bayesian optimization—by evaluating their performance on an LSTM model applied to a generated ARIMA-based financial dataset. The algorithms were assessed based on multiple criterias: mean-squared error, training time, convergence, generalization, and consistency.

This paper reveals that all three algorithms effectively identified similar hyperparameter configurations, each with its own strengths and trade-offs. The random search, despite its simplicity, achieved the lowest Mean Squared Error (MSE), highlighting its effectiveness in hyperparameter tuning. Hyperband demonstrated superior resource and time efficiency compared to the other two algorithms, though it showed some limitations in discovering the optimal configurations. Conversely, Bayesian optimization offered great consistency and a smooth learning process but proved to be more computationally expensive and required careful parameter tuning.

Interestingly, while Bayesian optimization was expected to deliver the best results due to its sophisticated approach, our study found that random search achieved the lowest MSE. This result can be attributed to the fact that Bayesian optimization relies on a surrogate model and acquisition function, which can be sensitive to initial conditions and parameter settings. In contrast, Random Search's broader exploration of the hyperparameter space allowed it to find particularly effective configurations, demonstrating that simpler methods can sometimes outperform more advanced algorithms, especially under limited computational resources.

Looking ahead, future research could explore several ways to build on these findings. For instance, developing optimization methods that integrate the strengths of random search, Hyperband, and Bayesian optimization might reveal new methods for achieving even better performance. Additionally, comparing these traditional methods to more novel algorithms such as genetic algorithms, gradient-based algorithms, and automated machine learning frameworks could provide insights into their relative effectiveness. Moreover, expanding the application of these algorithms to a more diverse range of datasets, including real-world financial time series or other fields could offer further insights into their generalizability and effectiveness.

References

1. Alibrahim, H.; Ludwig, S. A. Hyperparameter Optimization: Co mparing Genetic Algorithm against Grid Search and Bayesian O ptimization. In *2021 IEEE Congress on Evolutionary Computation (CEC)*; 2021; pp 1551–1559. https://doi.org/10.1109/CEC45853 .2021.9504761.
2. Amato, F.; López, A.; Peña-Méndez, E. M.; Vaňhara, P.; Hampl, A.; Havel, J. Artificial Neural Networks in Medical Diagnosis. *Jo urnal of Applied Biomedicine* **2013**, *11* (2), 47–58. https://doi.org/1 0.2478/v10136-012-0031-x.
3. *A review of stock market prediction with Artificial neural network (A NN) | IEEE Conference Publication | IEEE Xplore*. https://ieeexplo re.ieee.org/abstract/document/6720012.
4. *Review of State-of-the-Art Design Techniques for Chatbots | SN Com puter Science*. https://link.springer.com/article/10.1007/s42979-0 20-00255-3.
5. Hagras, H.; Pounds-Cornish, A.; Colley, M.; Callaghan, V.; Clark e, G. Evolving Spiking Neural Network Controllers for Autonom ous Robots. In *IEEE International Conference on Robotic and Auto mation, 2004. Proceedings. ICRA '04. 2004*; 2004; Vol. 5, pp 4620- 4626 Vol.5. https://doi.org/10.1109/ROBOT.2004.1302446.
6. Yu, Y.; Si, X.; Hu, C.; Zhang, J. A Review of Recurrent Neural Net works: LSTM Cells and Network Architectures. *Neural Computat ion* **2019**, *31* (7), 1235–1270. https://doi.org/10.1162/neco_a_11 99.
7. Terbuch, A. LSTM Hyperparameter Optimization: Impact of the Selection of Hyperparameters on Machine Learning Performanc e When Applied to Time Series in Physical Systems. Master's Th esis, 2021.
8. Yang, L.; Shami, A. On Hyperparameter Optimization of Machin e Learning Algorithms: Theory and Practice. *Neurocomputing* **20 20**, *415*, 295–316. https://doi.org/10.1016/j.neucom.2020.07.061.
9. Chollet, F., & others. (2015). Keras. GitHub. Retrieved from http s://github.com/fchollet/keras
10. Bergstra, J.; Bengio, Y. Random Search for Hyper-Parameter Op timization.
11. Li, L.; Jamieson, K.; DeSalvo, G.; Rostamizadeh, A.; Talwalkar, A. Hyperband: A Novel Bandit-Based Approach to Hyperparame

ter Optimization.
12. Frazier, P. I. Bayesian Optimization. In *Recent Advances in Optimization and Modeling of Contemporary Problems*; INFORMS TutORials in Operations Research; INFORMS, 2018; pp 255–278. https://doi.org/10.1287/educ.2018.0188.
13. Bergstra, J.; Bardenet, R.; Bengio, Y.; Kégl, B. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems*; Curran Associates, Inc., 2011; Vol. 24.
14. Dhake, H.; Kashyap, Y.; Kosmopoulos, P. Algorithms for Hyperparameter Tuning of LSTMs for Time Series Forecasting. *Remote Sensing* **2023**, *15* (8), 2076. https://doi.org/10.3390/rs15082076.
15. *Automated Machine Learning: Methods, Systems, Challenges*; Hutter, F., Kotthoff, L., Vanschoren, J., Eds.; The Springer Series on Challenges in Machine Learning; Springer International Publishing: Cham, 2019. https://doi.org/10.1007/978-3-030-05318-5.
16. Loshchilov, I.; Hutter, F. CMA-ES FOR HYPERPARAMETER OPTIMIZATION OF DEEP NEURAL NETWORKS. **2016**.
17. Ruder, S. An Overview of Gradient Descent Optimization Algorithms. arXiv June 15, 2017. https://doi.org/10.48550/arXiv.1609.04747.
18. Siami-Namini, S.; Tavakoli, N.; Siami Namin, A. A Comparison of ARIMA and LSTM in Forecasting Time Series. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*; 2018; pp 1394–1401. https://doi.org/10.1109/ICMLA.2018.00227.
19. Vincent, A. M.; Jidesh, P. An Improved Hyperparameter Optimization Framework for AutoML Systems Using Evolutionary Algorithms. *Sci Rep* **2023**, *13* (1), 4737. https://doi.org/10.1038/s41598-023-32027-3.
20. Box, G. E. P.; Jenkins, G. M.; Reinsel, G. C.; Ljung, G. M. *Time Series Analysis: Forecasting and Control*; John Wiley & Sons, 2015.

Author

Anon Taulananda is a junior high school student studying at Singapore International School of Bangkok. He is passionate about computer science and data science with a particular interest in machine learning. He wishes to pursue higher education at Massachusetts Institutes of Technology and Princeton university.

DOI: 10.36838/v6i9.2