Received: August 7, 2020. Revised: October 14, 2020. 212

INASS

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Online Tuning of Hyperparameters in Deep LSTM for Time Series Applications

Norah Bakhashwain[1] Alaa Sagheer[1,2]*

[1]*College of Computer Sciences and Information Technology, King Faisal University, Saudi Arabia*
[2]*Center for Artificial Intelligence and Robotics, Department of Computer Science, Aswan University, Egypt*
* Corresponding author's Email: asagheer@kfu.edu.sa

Abstract: Deep learning is one of the most remarkable artificial intelligence trends. It stands behind numerous recent achievements in several domains, such as speech processing, and computer vision, to mention a few. Accordingly, these achievements have sparked great attention to employing deep learning in time series modelling and forecasting. It is known that the deep learning algorithms built on neural networks contain multiple hidden layers, which make the computation of deep neural network challenging and, sometimes, complex. The reason for this complexity is that obtaining an outstanding and consistent result from such deep architecture requires optimizing many parameters known as hyperparameters. Doubtless, hyperparameter tuning plays a critical role in improving the performance of deep learning. This paper proposes an online tuning approach for the hyperparameters of deep long short-term memory (DLSTM) model in a dynamic fashion. The proposed approach adapts to learn any time series based application, particularly the applications that contain streams of data. The experimental results show that the dynamic tuning of the DLSTM hyperparameters performs better than the original static tuning fashion.

Keywords: Deep learning, Deep LSTM, Hyperparameter optimization, Online learning, Time series applications.

1. Introduction

Time series modelling and forecasting have drawn significant attention in various domains such as finance, engineering, and statistics [1]. Thus, many research papers have focused on algorithms and techniques that can yield accurate performance in numerous practical applications [2]. Many vital techniques have been proposed in the literature for improving the accuracy and efficiency of time series modelling and forecasting. The conventional statistical techniques of time series of modelling and prediction commonly use a potential model, such as autoregressive-moving average, autoregressive integrated moving average, and vector autoregressive to model and forecast using time series data [2-3].

However, these techniques have some concerns; one of them is requiring a deal with the whole dataset to identify the parameters of the model and give a response for each testing data [4]. This way, to identify the parameters, is not suitable in big datasets, mainly when a fast response is demanded in some applications [5]. Consequently, learning-based techniques attracted much attention as alternatives to conventional statistical techniques [5, 6].

Over the last two decades, machine learning techniques, mainly when applied to artificial neural networks (ANNs) have played an increasingly vital role in the design of computer vision and pattern recognition systems [7]. More precisely, it could be argued that the emergence of deep learning techniques has been crucial in the recent success of recognition and classification applications. In a sense, deep learning's success lies in capturing latent features in the input data, even if the size of the dataset is significant [8]. It also attempts to model high-level abstractions in the data by using multiple processing layers with complex structures [9].

Usually, the deep learning algorithms have several variables, called hyperparameters that should be selected in advance and before optimizing the model's parameters to be used in the validation phase [10]. These hyperparameters often tuned either manually or automatically using some hyper-learner algorithms. The automatic selection of suitable

hyperparameters is crucial to get optimal and faster results, primarily when large datasets are used. Of course, the essential factor in selecting the appropriate forecasting model. In other words, when the accuracy is high, this will help make accurate decisions and the opposite if we have low accuracy. In practice, to develop an efficient model shows an accurate performance is not an easy task, especially when the data is coming in a stream [11].

The second author of this paper proposed a deep learning algorithm, called deep long short-term memory (DLSTM), which used efficiently with time series forecasting problems [12]. DLSTM is an unsupervised learning model as an extension of the traditional recurrent neural network (RNN) model. It showed a significant improvement in the performance of time series prediction, where it adapts with learning the nonlinearity and complexity of time-series data [13]. DLSTM includes multiple LSTM layers such that each layer contains multiple neurons. In his paper, the second author showed that DLSTM demonstrates more effective use of the parameters of each LSTM's layer in order to train the forecasting model efficiently.

However, learning the parameters of an unsupervised architecture is quite tricky [14, 15]. These architectures often feature many hyperparameters that affect generalization performance, quickly creating a challenging tuning problem for human users [16].

Hyperparameters are essential for machine learning algorithms since they directly control the training algorithm's behaviours and have a significant effect on the performance of machine learning models. Therefore, in this paper, we are concerned about improving the performance of DLSTM through an online tuning for the hyperparameters of DLSTM. The proposed approach achieves the online tuning of hyperparameters by using the genetic algorithm (GA) dynamically. The experimental results show that the GA will determine the values of the optimum hyperparameters and, consequently, optimize the model performance and ensure the model's convergence.

This paper is organized as follows. Section II shows an overview of the related methods of solution and our motivation to propose our solution. Section III provides the details of the proposed research method and the experiments that we conducted. Section IV shows the results and their corresponding discussion. Section V concludes this paper.

2. Related methods and motivation

Though deep neural networks (DNNs) algorithms, such as DLSTM, dominate the modern machine learning landscape, their training and success still suffer from sensitivity to empirical choices of hyperparameters such as model architecture, loss function, and optimization algorithm. Let us define the hyperparameters and distinguish between them and the model parameters. The model parameters are those variables estimated from the input data "automatically". The model uses them to predict unseen data [11]. Examples for the model parameters in ANN are the weight coefficients and bias.

On the other hand, the hyperparameters are those variables, which we provide their values "manually" to the model to help it in the learning process and estimate the model parameters [11]. For example, the number of hidden neurons and layers, the learning rate, the activation function. These are examples of hyperparameters in ANNs; however, in other methods, they will be the kernel and slack in SVM, the value of k in k-NN and k-means, the number of levels in a decision tree. They will not appear in the final prediction, but they have a considerable influence on how the model parameters look after the learning step.

The tuning of the hyperparameters is not an easy task, particularly in time series applications where there is no rule of thumb. Nevertheless, there are some computational strategies and search approaches that address the optimization of hyperparameters. We can conclude them as follows,

1. **Grid search.** It is a brute force approach constrained by a pre-defined set of hyperparameters combinations, i.e., grid points. This strategy is feasible because the number of evaluations is lower than the brute force, and it allows reaching good results, as shown in [17]. However, the values of the hyperparameters must be defined manually or using a trial and error approach. This way requires much experience and remains a tedious task, even for experts. Besides, it is a computationally intensive method since the number of model evaluations grows exponentially with the number of hyperparameters [18].
2. **Random search.** It includes testing a pre-defined number of randomly sampled hyperparameters combinations where sampling is done uniformly at random [19]. It is widely shown that the random search about the hyperparameters' values is more efficient in practice than grid search. Typically, it leads to better-learned models

through less computation time. However, the random search cannot assure to find out the values that optimize the performance of the network [18].

3. **Bayesian approximations**. The positive side of this approach is that they do not have to entirely run the neural network to optimize it because they are grounded in an approximation [21]. This approach includes selecting the sampling points from which we can approximate the function in an intelligent way rather than selecting them randomly or on a grid [22]. However, the complexity of these approximations makes them close to being unfeasible and challenging to be parallelized. More precisely, Bayesian approximation has some hyperparameters that significantly affect its approximation performance, namely, choices of a non-convex acquisition function and the kernel. Moreover, maximizing such an acquisition function is required for each iteration of the approximation process [23].
4. **Evolutionary algorithms**. Similar to the Bayesian approximations, these algorithms seek in the search space of hyperparameters about those values that may optimize the performance of the model [24]. Nevertheless, the own definition of evolutionary algorithms has specific strategies for finding the right values in the search space. Moreover, these algorithms are parallelizable compared to Bayesian approximations; indeed, they are parallelizable in GPUs [9].

As we can see, many related methods have been employed for tuning the hyperparameters. In all of these studies, choosing the model hyperparameters represents a limitation, and most of the existing methods follow the manual adjustment of these hyperparameters. Therefore, an online tuning method for selecting the optimum hyperparameters is highly required for learning a DNN model. We believe that online tuning for deep models that solve the time series forecasting problem will improve prediction accuracy [24-25].

This paper will apply the online tuning for hyperparameters of the DLSTM model [12]. We will investigate how the dynamic learning approach of these hyperparameters will be beneficial to the static learning approach. To have an objective investigation, we selected two datasets, the first includes stock market data, and the other includes oil production data. We selected these datasets since the samples of both datasets are coming in reality as a stream. Therefore, online learning for such applications will be beneficial for getting a high performance.

3. Research method and experiments

This section shows all details about the proposed method and experimental settings.

A. The DLSTM model

The core concept of DLSTM is based on the notion that increasing the depth (or several hidden layers) of a neural network is beneficial for the overall performance [7]. The DLSTM is developed, as a deep architecture to the standard LSTM recurrent network, to be used in time series forecasting applications. Fig. 1 contains a stack of several LSTM blocks, one above another, and connected in a deep recurrent network fashion to combine the advantages of each LSTM layer.

As the authors explained in their paper [12], the goal of stacking several LSTM in such a deep architecture is to build the features at the lower layers that will disentangle the factors of variations in the input data and then combine these representations at the higher layers [12]. The DLSTM works as follows; each layer processes some part of the corresponding task and subsequently passes it up to the upper layer until the top layer eventually produces the output. In this way, such hierarchy allows the hidden layers at each level to run at a different timescale, which fits with time series applications. It is demonstrated that in case of big data applications, the deep architecture generalizes well in terms of the compact feature representation that not available in shallow architecture [26].

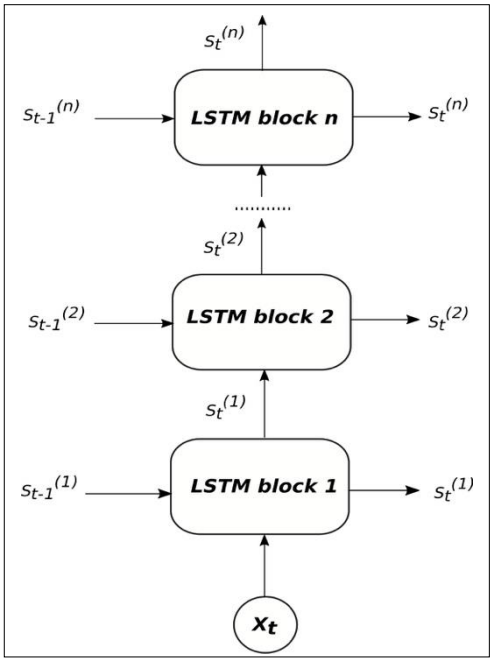

Figure. 1 The architecture of DLSTM model [12]

B. The proposed method

In the general machine learning domain, measuring the performance of a trained learning algorithm is performed in the validation phase using unseen data samples. To get excellent results of the machine learning model at hand, you should learn or tune the hyperparameters to find the best function [11],

$$f: X \rightarrow R \tag{1}$$

of a hyperparameter x $\in$ X. Therefore, the hyperparameters optimization aims to find the optimum global value of x of an unknown black-box function f(x) [10]. One of the main challenges of the neural network models, either shallow or deep, is finding the best tuning of the model hyperparameters, which is essential to the model's performance. While the earlier approaches that usually used for fine-tuning are either computationally unaffordable (grid search) or uncertain efficiency, e.g., trial and error like random search.

In this paper, we apply the GA to find the best hyperparameters of this DLSTM model [27-28]. Many applications show excellent performance when using a GA to solve complex optimization problems by selecting the right hyperparameter for the model. In this paper, we implemented the GA by using the library of distributed evolutionary algorithms in python [29]. We used a DEAP tools to define individual (since the chromosome is represented by a binary encoded string (10 bits), Beronoulli's distribution), then created the population, we used ordered crossover, after that we used mutShuffleIndexes mutation, then we used the roulette wheel selection for selecting the parents: the data used in this experiment has the following empirical parameters: the population size is 5, the number of maximum generations is set as 10, the number of genes= 1, the head length is 4. We have choosen the fitness function based on the mean absolute error (MAE) and root mean square error (RMSE).

Here, the implementation phase specifies how many hyperparameters we have to tune. Two scenarios are there, static scenario and dynamic scenario. In the static scenario, we identified three hyperparameters manually tuned, namely, number of epochs, number of hidden neurons, and the lag size [12]. While in the dynamic scenario, we specified four hyperparameters auto tuned by using the GA, where the first three hyperparameters are the same as these exist in the static scenario. The fourth hyperparameter is the number of times we update the forecasting model each time step when new observations from the testing data are inserted, named as the number of updates. Besides, the search for GAs to find the best solution is preformed over genetic structures that can represent several possible solutions. Table 1 shows the steps we followed to implement the GA in our experiments, where Fig. 2 represents these steps as a flow chart.

Table 1. The genetic algorithm procedure

```
1: Population ← [ list of n models on separate graphs]
2: Generation ← 0
3: While (generation < 1) do:
4:    train_and_evaluate (population)
5:    new_gen ← retain the m fittest individuals
6:    new_gen ← append random individuals to promote
        diversity
7:    mutate (new_gen)
8:    new_gen ← append offspring through crossover
        until k
9:    population ← new_gen
10:   generation < generation + 1
11: Outputs the hyperparameter of the fittest in
      population
```

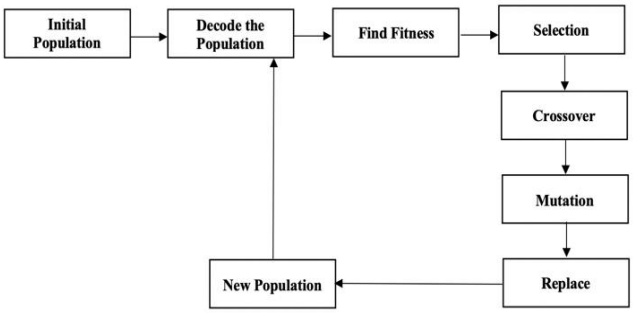

Figure. 2 A block diagram of the GA procedure

C. The learning algorithm

It is known that the neural network is converted to be an optimization problem that includes an objective function that requires either maximization or minimization concerning its parameters. Often, objective functions are stochastic needs to be optimally solved [30]. Usually, such stochastic objective functions are solved by calculating the stochastic gradient descent (SGD) optimization. SGD has a core practical importance in most kinds of ANN.

Adam optimization is an algorithm derived from adaptive moment estimation for efficient stochastic optimization that only requires first-order gradients with little memory requirement [31]. It uses estimations of the first and second moments of the gradient to adapt the learning rate for each parameter (or weight) of the neural network. In Adam, the update rule for individual weights is to scale their gradients inversely proportional to a scaled L2 norm of their current and past gradients. Another Adam

variant called Adamax, where the update rule is generalized using infinity instead of L2 norm [31]. In this paper's experiments, we use both algorithms, i.e. Adam and Adamax, and compare the overall performance.

D. Forecasting accuracy measure

To assess DLSTM [12] using the proposed approach, we conducted several experiments where some criteria measured the results. Two performance measures are employed; the first is the mean absolute error (MAE), as shown in Eq. (2). MAE is a standard measure in the forecasting field where it measures the average magnitude of the errors in a set of predictions, regardless of their direction. Therefore, it is the average over the test sample of the absolute differences between the prediction and actual observation, where all individual differences have equal weight [13]. The second measure is the root mean square error (RMSE), represented in Eq. (3). RMSE calculates the square root of the average squared differences between the actual and predicted observations [13].

$$\text{MAE} = \frac{\sum_{i=1}^{n} |y_i^{obs} - y_i^{pred}|}{n} \quad (2)$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (y_i^{obs} - y_i^{pred})} \quad (3)$$

E. Datasets description

Toward a fair assessment, two case studies from the industrial field and the financial field are employed as follows.

i. Case-1: Stock market price data

This case study dataset contains the samples of n = 41.266 minutes of data collected from Google Finance API to measures the performance of 500 companies on stock exchanges of the United States, ranging from April to August 2017 [32]. For this paper's experiments, we divided this dataset into two disjoint sets, 80% as a training set and 20% as a testing set.

ii. Case-2: Oilfield production data

The second case study concerns a dataset containing the sample Block-1 Huabei oilfield located in north China [12]. This oilfield was developed in 1992. The dataset contains observations of monthly data. The training set contains 85% of the sample, where the rest is used as a testing set.

F. Hardware and software platforms

All experiments of this paper are implemented on a workstation-PC equipped with Ubuntu 16.04 operating system. The system installed is Intel Core™ i7-6700 CPU @ 3.40GHz, 8.00 GB RAM, x64 based processor under python 2.7 software environment. For the DLSTM method, we used the source code developed by the second author in his paper [12], in which the Keras library was used with an open-source Tensor Flow library in the backend [33].

4. Results and discussion

We proceed now to show the quantitative and visual results of each case study using the DLSTM model after we apply the GA to select the hyperparameters and using both the Adam and Adamax optimizers. The original DLSTM model [12] is implemented in the static scenario, where our approach using the GA is called the dynamic scenario. Notably, in the static scenario, the experiments were repeated more than once by setting different values for the number of layers, the number of hidden neurons, and the number of epochs. The shown values in all tables are those who showed the lowest errors.

After that, the two-performance metrics have been applied to both training and testing data. The results in all the following tables indicate the performance of the model in the testing data rather than training data. We widely saw that the genuine evaluation for forecasting performance should be based on unseen data, not the historical data, which already seen by the forecaster [12].

a. Case-1: Stock market price data

Table 2 demonstrates the results of the first case study and the best hyperparameters values of the

Table 2. Results of DLSTM "Static-Adam" case 1

No. of layers	No. of epochs	No. of neurons	Lag size	RMSE	MAE
3	457	[5, 5, 2]	3	0.045	0.022
3	653	[1, 3, 3]	3	**0.022**	**0.021**

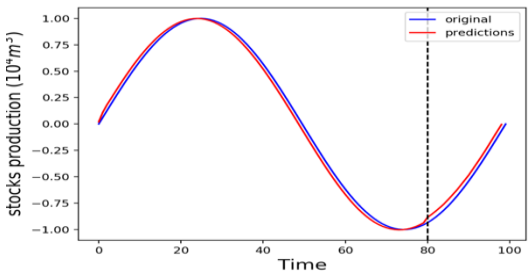

Figure. 3 Results of DLSTM "Static-Adam" case 1

Table 3. Results of DLSTM "Dynamic-Adam" case 1

No. of layers	No. of epochs	No. of neurons	Lag size	RMSE	MAE
3	787	[2, 2, 5]	5	0.007	0.007
3	953	[2, 2, 3]	1	**0.026**	**0.020**

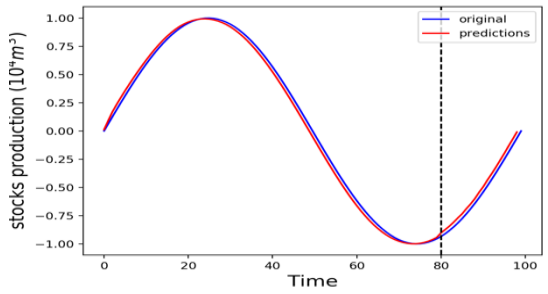

Figure. 4 Results of DLSTM "Dynamic-Adam" case 1

DLSTM model in the static scenario using Adam optimizer. Here, the table lists the values of the hyperparameters of DLSTM that we got through the trial and error approach. We call this approach a static approach. Fig. 3 shows the visual representation of the relation between the original stock price data and their prediction counterparts.

In contrast, Table 3 demonstrates the values of the hyperparameters of DLSTM using Adam optimizer. This approach we call it a dynamic scenario since the hyperparameters have been optimally selected using the GA. Fig. 4 shows the relation between the original data and their prediction in this scenario.

It is clear from the above results that the best results (lowest errors) are those attained in the dynamic scenario where the hyperparameters generated automatically using the GA. More precisely, the error values of the dynamic scenario are less than those of the static scenario. This means that the dynamic approach using the GA to auto-select the best hyperparameters of DSLTM using Adam optimizer improves the overall performance.

To confirm the dynamic approach's assessment, we use the "Adamax" optimizer instead of Adam optimizer. Table 4 shows the results of this assessment. It is clear that, with Adamax, the error values are decreased to (RMSE: 0.006 and MAE: 0.005), compared to both (RMSE: 0.007 and MAE: 0.007) in the dynamic -Adam scenario, and (RMSE: 0.007 and MAE: 0.007) in the static -Adam. Fig. 5 displays the relation between the original stock price data and their prediction through the dynamic scenario using the Adamax optimizer. It is clear from the figure that both presentations are quite close to each other, which confirms that the model is highly accurate. Table 5 concludes the comparison between the static and dynamic scenarios using both optimizers.

Table 4. Results of DLSTM "Dynamic-Adamax" case 1

No. of layers	No. of epochs	No. of neurons	Lag size	RMSE	MAE
3	634	[5, 2, 5]	5	0.010	0.009
3	953	[2, 5, 4]	5	**0.006**	**0.005**

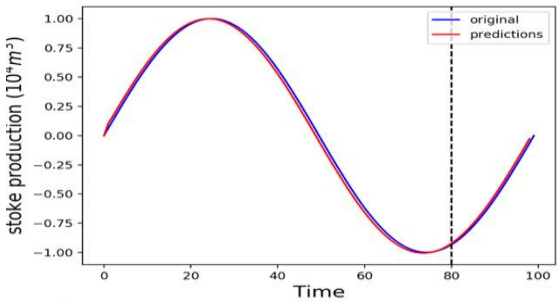

Figure. 5 Results of DLSTM "Dynamic-Adamax" case 1

Table 5. Overall comparison between static and dynamic scenarios using both optimizers case 1

Model	RMSE	MAE
Static-Adam	0.022	0.021
Dynamic-Adam	0.007	0.007
Dynamic-Adamax	**0.006**	**0.005**

Table 6. Number of epochs in all scenarios

	Static-Adam	Dynamic-Adam	Dynamic-Adamax
Number of epochs	**653**	787	953

Calculating the number of epochs required for the model convergence. We notice that the time consumed in the dynamic scenario, using both optimizers, is longer than the time consumed in the static scenario. Indeed, this is a natural attitude where the auto-generation of hyperparameter values takes time to find the optimum values. Overall, the online tuning for the hyperparameter improves the DLSTM performance; however, it comes on the account of computation time.

b. Case study 2: Oilfield production data

Table 7 presents the best error values and the best hyperparameters values of the DLSTM model in this case study. Here, the table lists the values of the hyperparameters that we got through the static approach using Adam optimizer. Fig. 6 shows the

Table 7. Results of DLSTM "Static-Adam" case 2

No. of layers	No. of epochs	No. of neurons	Lag size	RMSE	MAE
3	354	[4, 3, 5]	6	0.392	0.278
3	653	[4, 3, 5]	6	**0.286**	**0.237**

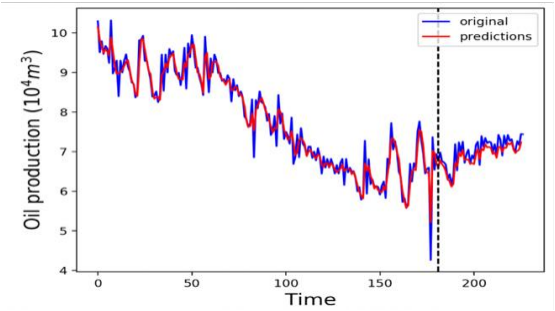

Figure. 6 Results of DLSTM "Static-Adam" case 2

Table 8. Results of DLSTM "Dynamic-Adam" case 2

No. of layers	No. of epochs	No. of neurons	Lag size	RMSE	MAE
3	634	[5, 2, 5]	5	0.316	0.248
3	**787**	**[2, 1, 2]**	**3**	**0.274**	**0.203**

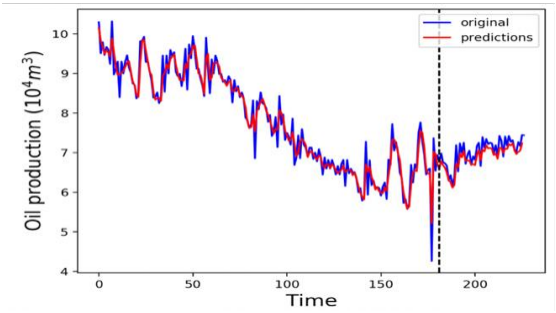

Figure. 7 Results of DLSTM, "Dynamic-Adam"-case 2

visual representation of the relation between the original production data and their prediction counterparts.

Table 8 shows the values of the hyperparameters that have optimally selected using the GA, i.e., the dynamic scenario, using the Adam optimizer. Fig. 7 illustrates the relation between the original production data and their prediction in this scenario.

Again, it is clear that the DLSTM through the dynamic scenario, where the hyperparameters are generated automatically using GA, outperforms its performance through the static scenario. In other words, the dynamic approach improves DLSTM model performance. This means that using the GA to auto-select the best hyperparameters using Adam optimizer enhances the overall performance.

In this experiment, we also use the "Adamax" optimizer instead of Adam optimizer to validate this improvement. Table 9 shows the results of this validation. It is clear that, with Adamax, the error values are decreased to (RMSE: 0.264 and MAE: 0.196), compared to both (RMSE: 0.274 and MAE: 0.203) in the dynamic -Adam scenario, and (RMSE: 0.286 and MAE: 0.237) in the static -Adam. Fig. 8 displays the relation between the original oilfield production data and their prediction through the dynamic scenario using the Adamax optimizer.

Table 9. Results of DLSTM "Dynamic-Adamax" case 2

No. of layers	No. of epochs	No. of neurons	Lag size	RMSE	MAE
3	787	[5, 1, 5]	3	0.304	0.234
3	787	[2, 2, 5]	5	**0.264**	**0.196**

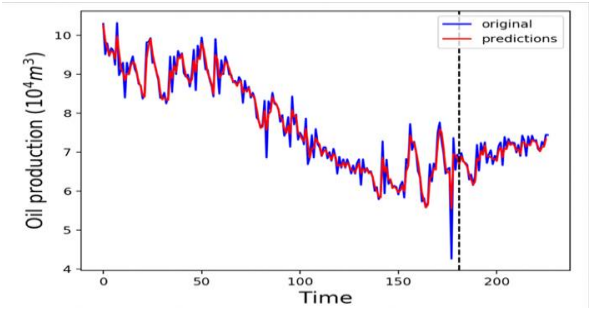

Figure. 8 Results of DLSTM "Dynamic-Adamax" case 2

Table 10. Overall comparison between static and dynamic scenarios using both optimizers case 2

Model	RMSE	MAE
Static	0.286	0.237
Dynamic – Adam	0.274	0.203
Dynamic-Adamax	**0.264**	**0.196**

Table 11. Number of epochs in all scenarios

	Static-Adam	Dynamic-Adam	Dynamic-Adamax
Number of Epochs	**653**	787	787

Again, it is clear that both data presentations typically close to each other, which newly confirms the model efficiency. Table 10 concludes the comparison between the static and dynamic scenarios using both optimizers.

Table 9 shows the estimated time for each scenario by calculating the number of epochs required for the model convergence. Newly, the time consumed in the dynamic scenario, using both optimizers, is longer than the time consumed in the static scenario. This confirms that the online tuning for the hyperparameter improves the overall performance, but it comes on the account of computation time.

5. Conclusion

Recent advances in deep learning have sparked great attention in order to employ deep learning in time series modeling and forecasting applications. However, tuning the hyperparameter of a model plays a vital role in the deep learning computation and enhances the overall performance. In this paper, we proposed an online tuning approach for the deep LSTM model presented before by the second author.

The proposed approach examined to tune the hyperparameters of two-time series case studies, namely, stock market price dataset and oilfield production dataset. Both case studies contain streams of data every second, making them more suitable to assess the proposed online tuning approach. The experimental results showed that the dynamic tuning of the hyperparameters performs better than the deep model's static tuning. In this paper's future work, we will try other optimizers, which developed recently, overcoming the current optimizer limitations.

Conflicts of Interest

The authors declare that there is no conflict of interest.

Author Contributions

Conceptualization, A.S.; Methodology, N.B., A.S.; Software and Experiments, N.B.; Validation, N.B. and A.S.; Formal Analysis, N.B. and A.S.; Data Curation, N.B.; Supervision A.S.; Writing—Original Draft Preparation, N.B and A.S.; Review & Editing A.S.

References

[1] D. Montgomery, C. Jennings and M. Kulahci "Introduction to time series analysis and forecasting", *John Wiley and Sons*, 2015.

[2] S. Athiyarath, M. Paul and S. Krishnaswamy, "A Comparative Study and Analysis of Time Series Forecasting Techniques", *SN Computer Science*, Vol. 1, No. 3, 2020.

[3] A. Parmezan, V. Souza, E. Gustavo, and P. Batista, "Evaluation of statistical and machine learning models for time series prediction: Identifying the state-of-the-art and the best conditions for the use of each model", *Info. Sci*. pp. 484: 302–337, 2019.

[4] Y. Haimi, P. Zhisong, and T. Qing, "Robust and Adaptiv Online Time Series Prediction with Long Short-Term Memory", *Computational Intelligence and Neuroscience*, Vol. 16, 2017.

[5] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, "Statistical and Machine Learning forecasting methods: Concerns and ways forward", *PLOS ONE, Public Library of Science*, Vol. 13, No. 3, pp. 1-26, 2018.

[6] G. P. Zhang, "Neural Networks for Time-Series Forecasting", *In: G. Rozenberg, T. Bäck, JN. Kok (eds) Handbook of Natural Computing*, pp. 461-477, 2012.

[7] J. Schmidhuber, "Deep learning in neural networks: An overview", *Journal reference: Neural Networks*, Vol. 61, pp. 85-117, 2015.

[8] Q. Zhang, LT. Yang, Z. Chen, and P. Li, "A survey on deep learning for big data", *Information Fusion*; Vol. 42, pp. 146–157, 2018.

[9] Y. Bengio, "Learning deep architectures for AI. Found. Trends Machine Learning", Vol. 2, No. 1, pp. 1–127, 2009.

[10] M. Feurer, F. Hutter, L. Kotthoff, and J. Vanschoren, "Hyperparameter Optimization", *In: (eds) Automated Machine Learning, The Springer Series on Challenges in Machine Learning, Springer, Cham*, 2019.

[11] G. Peter, M. Matskevichus, "Hyperparameters Tuning for Machine Learning Models for Time Series Forecasting", *Sixth International Conference on Social Networks Analysis, Management and Security (SNAMS)*, Granada, Spain, pp. 328-332, 2019.

[12] A. Sagheer and M. Kotb, "Time Series Forecasting of Petroleum Production using Deep LSTM Recurrent Networks", *Neurocomputing*, pp. 323:203-213, 2019.

[13] A. Sagheer and M. Kotb, "Unsupervised Pre-training of a Deep LSTM-based Stacked Autoencoder for Multivariate Time Series Forecasting Problems", *Scientific Reports, Nature*, Vol. 9, No. 19038, 2019.

[14] H. Larochelle, M. Mandel, P. Michael, and Y, Bengio, "Learning Algorithms for the Classification Restricted Boltzmann Machine", *The Journal of Machine Learning Research*, pp. 13:643–669, 2012.

[15] I. Goodfellow, M. Mirza, A. Courville, and Y. Bengio, "Multi-prediction deep Boltzmann machines", In: C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger (eds.) *Advances in Neural Information Processing Systems*, pp. 548–556, 2013.

[16] A. Ororbia, D. Reitter, J. Wu, and C. Giles. "Online Learning of Deep Hybrid Architectures for Semi-supervised Categorization", In: A. Appice, P. Rodrigues, V. Santos Costa, C. Soares, J. Gama, A. Jorge (eds) Machine Learning, and Knowledge Discovery in Databases. ECML PKDD, *Lecture Notes in Computer Science*, Vol. 9284, 2015.

[17] S. Mischa, S. Shahd, G. Julia, J. Tobias, N. Sebastien, S. Anett, "On the Performance of Differential Evolution for Hyperparameter Tuning", In: *Proc. of International Joint Conf. on Neural Networks (IJCNN)*, 2019.

[18] A. Sánchez-Il, D. Pérez-Guaita, D. Cuesta-García, J. D. Sanjuan-Herráez, M. Vento, J. L. Ruiz-Cerdá, G. Quintás, J. Kuligowski, "Model

selection for within-batch effect correction in UPLC-MS metabolomics using quality control - Support vector regression", *Anal. Chim. Acta*, pp. 1026:62-68, 2018.

[19] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization", J. Mach. Learn. Res. 13, pp. 281-305, 2012.

[20] K. Ozan and A. Barıs, "Nonlinear time series forecasting with Bayesian neural networks", *Expert Systems with Applications*, Vol. 41, No. 15, 2014.

[21] Y. Ozaki, M. Yano, and M. Onishi, "Effective hyperparameter optimization using Nelder-Mead method in deep learning", *IPSJ T Comput Vis Appl*, 2017.

[22] W. Jia, C. Xiu-Yun, Z. Hao, X. Li-Dong, and D. SiHao, "Hyperparameter Optimization for Machine Learning Models Based on Bayesian Optimization", *Journal of Electronic Science and Technology*, Vol. 17, 2019.

[23] T. Liu and J. Plested, "Using Evolutionary Algorithms for Hyperparameter Tuning and Network Reduction Techniques to Classify Core Porosity Classes Based on Petrographical Descriptions", In: *T. Gedeon, K. Wong, M. Lee (eds) Neural Information Processing, Communications in Computer and Information Science*, Vol. 1142, 2019.

[24] L. Viktor, H. Barbara, W. Heiko, "Incremental on-line learning: A review and comparison of state-of-the-art algorithms", *Neurocomputing*, Vol. 27531, pp. 1261-1274, 2018.

[25] S. Qing, Z. Xu, F. Haijin, and W. Danwei, "Robust Recurrent Kernel Online Learning", *IEEE Transactions On Neural Networks and Learning Systems*, Vol. 28, No. 5, 2017.

[26] M. Hermans and B. Schrauwen, "Training and analyzing deep recurrent neural networks", In: *Proc. of the 26th International Conf. on Neural Information Processing Systems NIPS 1*, pp. 190–198, 2013.

[27] R. Andonie, "Hyperparameter optimization in learning systems", *J Membr Comput*., Vol. 1, No. 4, pp. 279–291, 2019.

[28] C. Francescomarino, M. Dumas, M. Federici, G. Ghidini and L. Simonetto, "Genetic algorithms for hyperparameter optimization in predictive business process monitoring", *Information Systems*, Vol. 74, No. 1, pp. 67-83. 2018.

[29] J. Kim and S. Yoo, "Software review: DEAP (Distributed Evolutionary Algorithm in Python) library", *Genet Program Evolvable Mach*, Vol. 20, No.1, pp. 139–142, 2019.

[30] G. Villarrubia, J. De Paz, P. Chamoso, and F. Dela, "Artificial neural networks used in optimization problems", *Neurocomputing*, Vol. 27210, pp. 10-16, 2018.

[31] D. Kingma and J. Ba, "Adam: A method for stochastic optimization", In: *Proc. of the 3rd International Conf. on Learning Representations (ICLR)*, 2015.

[32] J. Sigel and D. Schwartz, "Long-term returns on the original S&P 500 companies", *Financial Analysts Journal*, Vol. 62, No. 1, pp. 18-31, 2006.

[33] M. Abadi, A. Agarwal, and P. Barham, "TensorFlow: Large-scale machine learning on heterogeneous systems", http://tensorflow.org/, 2015.

[34] M. Ivanović and V. Kurbalija, "Time series analysis and possible applications", *39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pp. 473-479, 2016.

[35] Z. Hajirahimi and M. Khashei, "Hybrid structures in time series modeling and forecasting: A review", *Engineering Applications of Artificial Intelligence*, Vol.86, pp. 83-106, 2019.