Accuracy improvement of Deep neural Networks through preprocessing and neural structure tuning techniques. An approach to time series models

Directors: Portela González, José
Pizarroso Gonzalo, Jaime
Author: López-Tafall Criado, Mónica

Resumen **—** La popularización de los modelos de la Red Neural Profunda en los últimos años ha dado lugar a un aumento del número de técnicas para aumentar la precisión de esos modelos. Sin embargo, es difícil elegir cuál de estos métodos son óptimos para un problema específico, así como lograr valores óptimos de sus hiperparámetros correspondientes para lograr un modelo de calidad. Este proyecto tiene por objeto proporcionar una comparación de éstos aplicándolos en un problema de predicción de series temporales.

Se han llevado a cabo una serie de test en los que se han combinado arquitecturas tipo RNN, GRU, LSTM y BiLSTM con métodos de optimización entre los que se han destacado el Grid Search, Random Search, Optimización Bayesiana y Algoritmos Genéticos.

Los resultados muestran que, se debe optar por una configuración tipo Grid Search + Red tipo GRU, la cual alcanza un valor de MSE de 167,60, frente al 240,44 del modelo benchmark o configuración tipo Random Search + RNN si se busca rapidez de ejecución, sacrificando precisión.

Abstract **--** The popularization of Deep Neural Network models in recent years has led to an increase in the number of techniques to increase the accuracy of these models. However, it is difficult to choose which of these methods are optimal for a specific problem, as well as to achieve optimal values of their corresponding hyperparameters to achieve a quality model. This project aims to provide a comparison of these by applying them to a time series prediction problem.

A series of tests have been carried out in which RNN, GRU, LSTM and BiLSTM type architectures have been combined with hyperparameter optimization methods among which Grid Search, Random Search, Bayesian Optimization and Genetic Algorithms have been highlighted.

The results show that, one should opt for a Grid Search + GRU type configuration, which reaches an MSE value of 167.60, compared to 240.44 in the benchmark model or Random Search + RNN type configuration if one seeks speed of execution, sacrificing precision.

I. Introduction

Future trends or behaviors of stock markets, product sales, electricity demand, wind speed and sun-irradiation for power generation, health- related issues and Natural Language Processing (NLP), among many more, have, for many decades (late 20th Century), been at the forefront of development of machine learning algorithms and ever evolving complex mathematical models [1].

Most of these prediction models' approach is to use machine learning algorithms to learn from the past in order to provide a future time-window forecast, although this is easier said than done. Due to the dynamic, chaotic, stochastic, and complex environments of the problem itself and uncertainty of real-world data (e.g. stock markets) this becomes an inherently challenging and non-trivial process. This gets worse when dealing with time-dependent characteristics or long-term multi-variate information chains, where it is of utmost importance to identify correlations between distinct temporal data [2] or to define which past values (both in time and variables) will be considered within the prediction process [3], respectively.

This work focuses exclusively on the study of Deep Neural Networks (DNNs) architectures, emerged as a means of mimicking the biological, complex behavior of the human brain. A single neuron (Perceptron) does not perform well enough, but further development lead to the wiring of multiple artificial neurons (synapses), so that the output of one became the input of the next one, creating multiple layers that would manage to provide much more accurate results, and acquire a much higher resemblance to the human brain behavior. Over the course of time this process has been renewed, optimized, and further developed to try to boost the prediction accuracy, leading to a vast portfolio of prediction model architectures.

This portfolio includes examples such as the Multi-Layer Perceptron (MLP), Deep Learning NNs, Recurrent NNs, Long-Short Term Memory NNs (LSTM), Convolutional NNs, Recursive NNs, etc. whose application depends mainly on the data provided and expected output, although variations of their usage can be found in the available literature.

When facing a new time series forecasting problem, one of the greatest challenges is how to choose the most convenient architecture and tuning technique to obtain the best possible results. This choice is nontrivial, and its complexity depends in part on the resources available, including time. It may be that time is a key factor in the application and that the models, therefore, must be able to be executed under strict time requirements. Or you may simply want to start an initial search process for the most suitable hyperparameters with a fast and simple model and that allows you to fine tune them by means of more complex models afterwards, without getting lost in the complex initial search.

In addition to choosing the right architecture, comes the need to adjust the number of hidden layers, number of units, activation functions, choosing the correct optimizer, epochs, dropout, batch size, learning rate, number of iterations, and

many more hyperparameters whose tuning is not trivial nor immediate or dependent on the expected output [4].

In order to try to answer this question, a selection of deep learning models will be tested under the same uncertainty conditions to assess which on would produce the best option based on those requirements the prediction application might withstand.

II. STATE OF THE ART

A. *Input variable selection*

Variable or feature selection (FS) is based on different algorithms that seek to automatically select attributes from the available data that are most useful for the predictive problem we might be dealing with. This is one of the most critical steps for achieving a high degree of accuracy.

In recent years, resources such as Genetic Algorithms, Harmony Search, Temporal Memory search, attention modules, Paragraph Vector, etc. have emerged to improve feature selection to assess the use of relevant not correlated input data that bring value to model operation, and help with vanishing gradient problem when in need to handle long-terms dependencies.

Filter based FS algorithms use statistical metrics or techniques to filter features giving each data column a feature score and ranks them by their predictive importance.

As these methods depends highly on the relationship between variables (input-output), the metric should be chosen based on the type of variables with which the prediction model will work with later on. The following figure allows a better understanding of the most suitable statistics for each case.

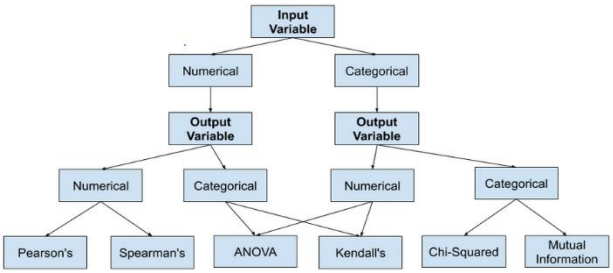

Figure 1. Filter based FS methods. Source:[5]

The main problem related to filter-based methods is that they are mostly univariate techniques, which means that each predictor is only evaluated regarding the output variable, discarding any possible interactions with other input variables. This, in turn, may lead to the training of the model with redundant, yet relevant, information, causing collinearity problems to appear.

A solution to test the interaction of several variables at the same time, *Wrapper-based FS algorithms* are used to measure the fitness of certain features based on how good or bad does the model perform after being trained with them and the level of generalization it can achieve, rather than obtaining intrinsic information from each variable in order to classify their relevance with respect to the output variable.

These methods are based in an iterative process in which different combinations of variables are prepared, evaluated and modified to come up with the optimal feature combination that best fits the prediction problem.

Some of the most widely used Wrapper based FS algorithms are *Genetic Algorithms* - The optimization process is done by allowing a population of individuals to evolve by randomly subjecting them to actions similar to those that act in biological evolution (genetic mutations and recombinations) [6]- , *Harmony search* – Inspired by the way in which musicians improvise a harmony by trying different combinations of pitches they know from experience (memory) and adjust the pitch of each instrument until they obtain the harmony they were looking for [7]- , *Temporal Memory Search* - Tries to identify the mount of temporary memory needed to solve the issue, being the memory the past time values (lags) of the time series [3]- ,. *Recursive Feature Elimination (RFE)* - in which an initial set of variables is trimmed gradually until only the most relevant variables are left, based on a coefficient or feature importance attribute, or until the desired number of relevant features is achieved -, or *Sequential Feature Selection* - an initial empty feature set is competed after each iteration with a number of variables until there is no improve in accuracy for the trained model-.

In addition to these, Embedded FS algorithms use algorithms to penalize features which coefficients are too high in order to reduce complexity and avoid over-fitting or variance of a model by adding extra bias. In order to achieve this, regularized methods, such as L1 regularization (Lasso), L2 regularization (Ridge Regression) or decision tress, are used to control the size of features' weights[8].

B. *Hybrid Models*

Neural Networks´ (NNs) different architectures aim to provide a wide range of options in order to target issues of multiple natures, such as image/video classification, forecasting, speech recognition, natural Language Processing (NLP), among others.

As interest in NNs' grew, so did the number of possible algorithms to choose from in search of improving performance accuracy. Most recent developments propose Hybrid architectures, that is models that combine multiple neural architectures such as LSTM + attention modules, Convolutional NNs' + RNNs', Deep Belief Networks, etc. Some of the most interesting hybrid models are:

- *Deep and Wide Neural Networks (DWNN)* [2] are a newly developed NN architecture in which a Convolutional layer is added to the hidden state of a Recurrent Neural Network. This way, the model not only accounts for the depth provided by RNNs' (time dimension, in the form of number of time lags), but also the width associated to CNNs' (variable number of data sets). This CNN layer is formed by convolution layers, a pooling layer and a full connection one in order to adapt the output into the required shape of a hidden state.

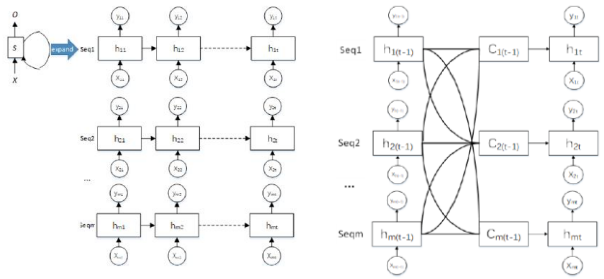

Figure 2. Simple RNN mode VS DWNN model. Source:[2]

This type of architectures have proven to achieve a good reduction in MSE obtained values of approximately 30% when compared to general RNN models [2].

- *TreNets* are hybrid Neural Networks architectures for the trend prediction of time series. This hybrid architecture combines long short-term memory (LSTM), a convolutional neural network (CNN), and a feature fusion layer. In this case, the LSTM network contains and passes on the historical trends that contain the long-term contextual information of the time series. This historic information is relevant as it may naturally affect the trend evolution. Alternatively, raw local data serves as an input for the CNN, which will extract useful information about the local behavior of the time series in order to determine the dependency of the current trend and pattern transition point. This kind of information can be of great relevance when predicting abruptly changing trends.

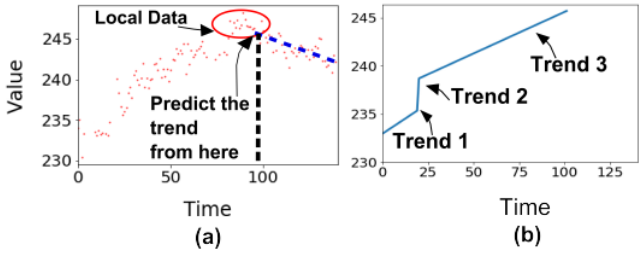

Figure 3. Local and historic trend data. Source:[9]

TreNets have shown to outperform simple LSTM-CNN models, reducing by 30% the achieved error (RMSE) at the maximum [10].

- *Bi-Directional LSTM + CNN layer models [11]* seek to achieve a higher degree of accuracy while reducing the probability of overfitting. Bi-directional LSTM networks are very useful when dealing with long spanning time-series data as they are able to identify key behaviors from both backwards and forward time dependencies. This provides the network a better understanding of the context which, in turn, accelerates the learning process. In order to reduce variance when encoding properties of input into the network, CNNs can be used prior to the execution of bi-directional LSTMs in order to facilitate feature extraction. The pooling layer present in CNNs also helps in achieving a more accurate relevant feature extraction by limiting variance due to small local distortions and reducing the feature space dimensionality.

This model claims to have increased accuracy by 9% with regards to a single pipeline CNN- Bi LSTM model [11].

- *Low-High CNNs* are a novel architecture that combines multi CNNs with multistep Attention modules [12]. In this case, various sets of CNNs are used for extracting (1) Low Level features and (2) High level features. Low level features are made up of local behaviors in different time steps (curves, onwards or downwards slopes, etc.) whereas High level features are built on top of these and extract larger shapes in temporal data.

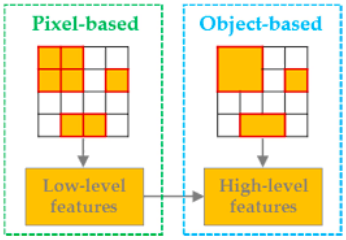

Figure 4. Low-High Level features in an image. Source: [13]

The attention module matches both outputs by identifying relevant context, just as it would be used in NLP for sentence translation. This enables to get better understanding of the problem context and helps in assessing how different data points related to each other without losing the temporal component.

C. Hyperparameter tuning

Once a neural architecture has been chosen to be trained, a variable number of hyperparameters must be tuned in order to achieve the best possible model with that architecture. These hyperparameters define how the network functions and are key to their validity and accuracy. Their values depend ultimately on the problem that is being addressed, type of available data and expected output. Moreover, they and are correlated among them, which means that any modification of a single hyperparameter might force to modify the rest.

Some of the most common or important hyperparameters to adjust within a NN are [14]:

- *Number of hidden layers*: Increasing the number of hidden layers is usually believed to increase model accuracy.
- *Cell units (neurons) per layer*: Same as with the number of hidden layers, a greater number of neurons per layer might help to optimally identify relevant behavior in data, as more interactions between variables can be taken into account. Having a large number of neurons might lead to an increase in computational weigh and even overfitting.
- *Parameter initialization*: It is necessary to initialize the weights in the first pass. These values can be set to zero or obtained with a random function. This can lead to vanishing or exploding gradients, which reinforces the need to find a way of easily initializing them without compromising its training.
- *Learning rate:* It represents the amount weights are increased during training. It usually has a positive value from 0

to 1. This value is one of the most relevant ones, as it can lead to heavy and long training process after which the process could get stuck (low values) or too fast and cause the training process to become unstable and achieve sub-optimal sets of weights (large values)[15]

- *Loss function*: it is the function designed to calculate the distance between the predicted output and the expected output. After computing the loss score, a learning algorithm (usually Gradient Descent) is then used to update the weights in a way that might achieve a better loss score in the next iteration.
- *Epochs, iterations and batch size*: These three hyperparameters define the way in which data is fed to the model. As explained in [16]:
  - An *epoch* is a forward pass and a backward pass of *all* the data samples in the training dataset.
  - The *batch size* is the number of data samples in each forward or backward pass. This value is set when the number of variables is too large to be run in one single epoch, or when the complexity of the model makes it necessary.
  - The *number of iterations* is the number of backward and forward passes using the information contained in each batch.
- *Dropout regularization*: Defines the number of neurons not trained in each epoch in order to avoid overfitting of the model during training.
- Optimizer algorithm and momentum: The neural network optimizer is in charge of running gradient descend in order to actualize the weights of each variable. The way to do so varies from one optimizer to another which can impact the model performance.

Although there is no straight-forward way to fine tune them, there are some methods that can facilitate this task and make it less complex, among which four stand out [14], [17], [18]:

- *Hand tuning* is the simplest and most straightforward of them all. This method can be used whenever knowledge and previous experience supports it, but it is not a scientific method and does not account for under optimized hyperparameters
- *Grid Search* is based in an iterative process that tries multiple values for each hyperparameter. These values can be either predefined or sorted out from an interval with a fixed step size and the model will train the model for each possible value and return its associated loss score. This method provides a means of mapping the problem space and more optimization capability.
- *Random Search* Random Search methods are strictly linked to Grid Search methods but only try randomized values of hyperparameters [17]. These values are gathered from the entire problem space, rather than form just promising areas that could hide a local optimum.

One of the main issues with this method is that it can sometimes leave some space points uncovered and it evaluates points that are too close to each other to really make a significant improvement. In order to avoid this, quasi-random sequences (also known low-discrepancy sequences) help to spread more evenly the data points. Some of these quasi-random sequences are the Sobol, Harmmersley, Halton, Kronecker and Niederrelter sequences [18], [19]

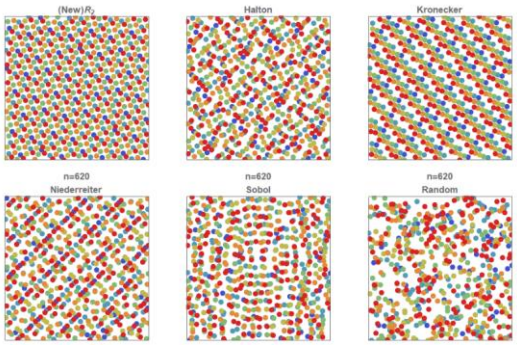

Figure 5. Comparison of some quasi-random search methods. Source: [19]

- *Sequential model-based optimization methods* represent a substantial improvement compared to Grid and Random search methods as they base their operation in trying to improve past evaluations through a probabilistic model, making assumptions about unobserved values through an Acquisition Function. The Acquisition Function is a surrogate of the original loss function of the model, making evaluations much simpler and faster. It can be built using either Gaussian Process, Random Forest or Tree Parzen Estimators (TPE).

These 4 methods retrieve most of the attention, but there are also alternative resources that may be used for hyperparameter optimization. Gradient-based optimization, Evolutionary Algorithms and Population-Based Algorithms are some of these alternative methods.

- *Gradient-based optimization methods* apply reverse stochastic descend methods with momentum to evaluate gradient descend related to each hyperparameter. Then, just as it is done with the weight's modifications, values for each hyperparameter are upgraded in order to obtain a better loss score for the model.
- *Evolutionary Algorithms* for hyperparameter tuning work just as described for Genetic Algorithms in section 1.3. These methods are widely used specifically for neural networks optimizations as they are easy to compute and perform well with various architectures [20]
- *Population-Based Algorithms* are a combination of both parallel search methods (Grid and Random Search) with Sequential Search methods (Hand and Bayesian optimization). It jointly learns both hyperparameters and networks weights, conducting a wholesome optimization of the prediction model [21].

III. CASE STUDY

Whereas Section II has served as an introduction to different prediction models and techniques that could be used to forecast time series values, it still remains unclear which would be most ideal to work with.

In order to face this, some of the described techniques will be tested under the same circumstances and for the same prediction objective to assess their overall performance and try to obtain valuable insights that could help when facing a new prediction problem

Certain assumptions have been made under which the development of this work is encompassed. The first one is that this work does not pretend to achieve a network configuration as optimal as possible, but to test how the network prediction responds to changes in the observed hyperparameters. It is expected that this will allow to derive conclusions about how the modification of each hyperparameter affects the response of the models with the aim of extracting relevant conclusions that can be applied to similar prediction problems, as well as identifying the most interesting algorithms according to the needs of the problem.

The other hypothesis of this work is that it is assumed that most of the results extracted can be reproduced in similar prediction problems. While it is understood that the answer cannot be 100% reproducible, it is expected that a high degree of similarity in the answer will be achieved to support the use of the conclusions derived from this work.

A. Software settings

All models have been developed in Python using Google Collab as the development environment. The choice of using Google Colab was made based on the computing requirements to run these models and usage simplicity, as no local install is required.

Google Colab allows to develop code on Jupyter notebooks and run it on Google's cloud servers using VPCs. It also provides access to additional computing resources dedicated to hardware acceleration, such as GPUs, including Nvidia K80s, T4s, P4s and P100s.

All models have been implemented using Tensorflow and Keras, the two most widely used frameworks nowadays for Machine Learning oriented applications.

B. Data set

The dataset used for testing the different techniques was obtained from a national wind mill park. It consists of 122 different variables and 35136 total hourly records that can be divided into 4 main separate type of inputs:

- T – Temperature in Celsius Degrees.
- GSR – Ground Solar Radiation
- WS – Wind Speed in m/s
- WD – Wind direction, in degrees to north

In order to train and validate the models, a 70:30 split ratio has been defined.

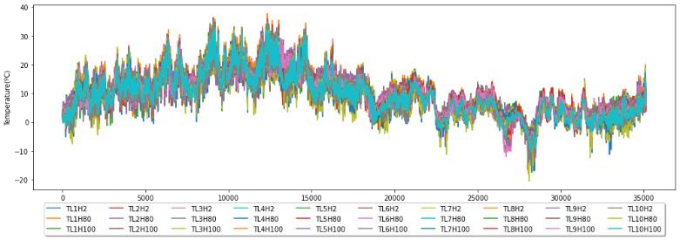

Figure 6. Temperature (ºC) variables. Min. = -18,7 ºC, Max. = 37,5 ºC

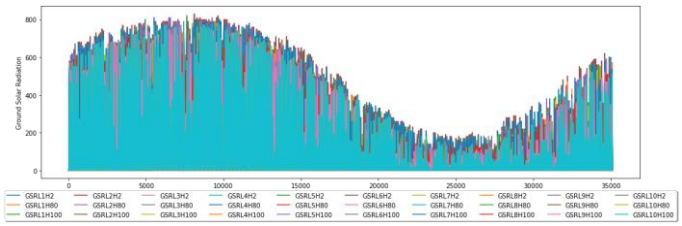

Figure 7. Ground solar radiation

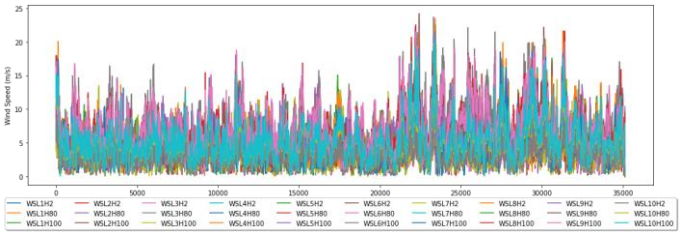

Figure 8.Wind speed (m/s)

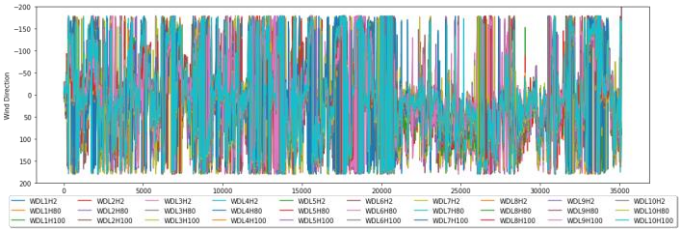

Figure 9.Wind direction (º)

The 5th variable, Wind Power Generation (WG), is the objective output of the model. The model should forecast its value in the next hour.

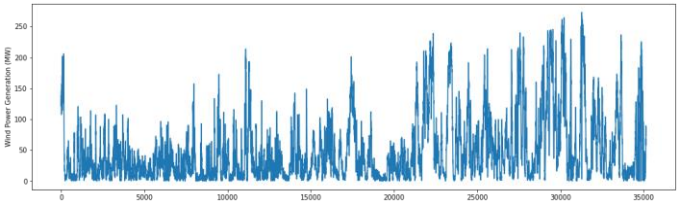

Figure 10.Output variable detail

C. Model assumptions

This workstream has been focused on assessing the impact of 4 of them (algorithm, numbers of layers, number of neurons per layer, and leaning rate) and fixing the values of some additional ones based on their overall acceptance according to the consulted bibliography or execution simplicity, given the computing resources used for the development of the models.

- Models: ['lstm', 'gru', 'rnn', 'bilstm']
- activation : ['elu']
- layers : (1,6,1), min_layers, max_layers (not included), step_size
- neurons_per_layer :(1, 45, 5), min_neurons, max_neurons (not included), step_size
- learning_rate: (0.0005, 0.005,5), min_lr, max_lr (included), number_of_elements
- optimizer: ['adam']
- batch_size: [20]
- epochs: [20]

The rank of values to be tested in the variable hyperparameters have been randomly chosen and do not correspond to any previous conducted tests.

D. Benchmark Model

In this work, the benchmark model has been defined under the assumption that WG at $t+1$ will be equal to the previous recorded value (WG value at time step t).

$$y_{pred}(t) = y\,(t-1) \quad (1)$$

Validation MSE will be, in this case, the metric used to identify those models that beat the benchmark model assumption for the same prediction problem.

The Validation MSE for the benchmark model stands at 240.44. Any model achieving a lower validation MSE will be classified as valid for our forecasting problem, at least accuracy wise.

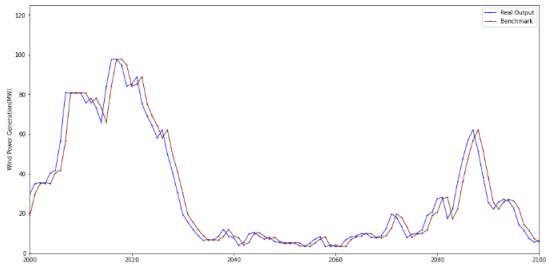

Figure 11. WG -Real output vs Benchmark model

IV. RESULTS

TABLE I
Cross comparison for the best performing simulations

	Tuning technique	Layers	Neurons per layer	Learning rate	Validation MSE	Total execution time (min)
LSTM	Grid	2	26	0,00158114	175,390069	260,21
LSTM	Random	3	31	0,00298181	184,265402	45,5
LSTM	Bayesian	3	15	0,00100075	182,650941	118,01
LSTM	Genetic Algorithm	3	30	0,00146076	187,247948	100,37
GRU	Grid	3	16	0,0005	167,600379	217,3
GRU	Random	4	26	0,00096535	169,507542	48,6
GRU	Bayesian	2	30	0,00063344	170,248371	100,14
GRU	Genetic Algorithm	5	39	0,00072304	175,601382	56
RNN	Grid	3	6	0,0005	179,856083	107,1
RNN	Random	1	21	0,00195347	191,678969	24,7
RNN	Bayesian	6	15	0,00073817	185,762487	70,17
RNN	Genetic Algorithm	4	86	0,00156522	176,363621	42
BILSTM	Grid	2	11	0,005	187,233101	511,6
BILSTM	Random	1	41	0,00414321	192,240764	97,3
BILSTM	Bayesian	1	45	0,00259197	190,741036	164,23
BILSTM	Genetic Algorithm	2	69	0,00229161	216,845031	144,07

TABLE II
Number of adjusted candidates per tuning technique

Tuning technique	Number of adjusted candidates
Grid Search	150
Random Search	50
Bayesian optimization	100
Genetic Algorithm	80

It can be seen how the best models are concentrated as a result of using GRU type network architectures as opposed to LSTM, as would have been expected based on the conclusions derived from the literature consulted, and how RNNs are positioned as the best architectures in terms of execution time, both results already presented in the previous sections.

LSTM architectures achieve acceptable results, although far from the best obtained with GRUs and without substantial improvements in terms of total execution time that could justify their choice when faced with a prediction problem similar to the one discussed. BILSTM architectures would be initially discarded for not being able to prove whether their low performance could be improved by means of other hyperparameters whose optimization is not contemplated in this work.

Regardless of the type of network architecture employed, it is clear how Grid Search techniques are the most interesting to use in a first approach to the problem. Not only are these techniques the ones that achieve, in a general way, the best results, but they also allow exploring the whole map of hyperparameters defined without being influenced by local minimums

As far as the speed of execution is concerned, - and in the same way for any network architecture - Random Search is the one that allows to obtain reasonable results in the shortest time possible, but, as it has been explained before, it is necessary to evaluate the need of greater resources to evaluate more complex configurations or to increase the number of iterations, which entails a great limitation to have in consideration for its implementation.

Bayesian optimization is an alternative solution if you are looking for an intermediate point between accuracy and execution time. Unlike genetic algorithms, which do not stand out in the field of accuracy or runtime improvement, the tests performed by Bayesian optimization are usually closer to the optimal that can be achieved through grid search, without the complexity of random and more feasible than tests performed by genetic algorithms that, after each new mutation, risk ending up selecting sub-optimal configurations.

V. CONCLUSIONS

Although it cannot be presumed that all prediction problems have the same behavior, nor that the models tested will respond in the same way to what has been seen in this work in other situations, it can be expected, and this has been assumed in this work, that it is possible to generalize some of the results obtained here, as long as one is aware of this fact and the limitations it may entail.

Another important aspect to take into account, and which has partly defined the results obtained, is that this work has only contemplated the optimization of the 4 hyperparameters that are most sought after in a general way when undertaking any work in this area, allowing a certain degree of freedom in the initialization of other more complex hyperparameters, whose theoretical basis does not allow direct conclusions to be extrapolated from the values obtained due to their complexity. Assumptions have also been made regarding the values of some hyperparameters that may be far from optimal, but which have greatly facilitated the development of this work.

In order to improve the results obtained in this work, it would be of great interest to carry out tests along the same lines of this work, testing other ranges of values for the hyperparameters tested here, with the aim of observing whether the behavior remains in accordance with what has been observed in this work, or whether conclusions can be drawn that could complement those presented here.

As for the results, the following points can be concluded:

- Regardless of the optimization technique employed, GRU architectures far exceed the expectations placed on LSTMs that were initially positioned as those from which to expect the best results, while biLSTMs are relegated to last position by failing to achieve good results in any of the fields tested.
- As far as possible, start the search for hyperparameters in limited environments and opt for intermediate values, avoiding any extreme values. As can be seen from the final table of results (Table 15), most of the results oscillate around 2-3 layers, a number of neurons per layer not exceeding 30 (with some exceptions) and with downward values as far as learning rate is concerned.
- The number of layers and neurons per layer should be adapted to each application, being aware that for more complex applications a greater number of layers and neurons would be required to model the output behavior. It is recommended to perform a more comprehensive initial search, and limit subsequent trials in a limited rank around those values within the optimal configuration appears to be located.

In order to choose the best hyperparameter optimization method, the following approach can be followed:

- *Is time a decisive factor?*

Opt for Bayesian optimization or random search, if you have sufficient computational resources to achieve a sufficient number of iterations, along GRU or RNN-type network structures, depending on whether you want better accuracy or just speed of execution, respectively. As for the number of iterations, it is necessary to at least execute a number of them that allows the algorithm to converge in a hyperparamenter subspace, ensuring a small variance from one iteration to the next one.

- *Is the best possible accuracy being sought?*

If, above all, the aim is to minimize the model's error, opt for Grid search optimization methods together with GRU-type network architectures.

- *The objective is to achieve a time-performance ratio?*

Opt for a Bayesian optimization method using GRU or RNN-type network architectures

REFERENCES

[1] H. Wan, 'Deep Learning:Neural Network, Optimizing Method and Libraries Review', in *2019 International Conference on Robots & Intelligent System (ICRIS)*, Haikou, China, Jun. 2019, pp. 497–500, doi: 10.1109/ICRIS.2019.00128.

[2] R. Zhang, Z. Yuan, and X. Shao, 'A New Combined CNN-RNN Model for Sector Stock Price Analysis', in *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, Tokyo, Japan, Jul. 2018, pp. 546–551, doi: 10.1109/COMPSAC.2018.10292.

[3] I. Valenca, T. Ludermir, and M. Valenca, 'Hybrid Systems to Select Variables for Time Series Forecasting Using MLP and Search Algorithms', in *2010 Eleventh Brazilian Symposium on Neural Networks*, Sao Paulo, Oct. 2010, pp. 247–252, doi: 10.1109/SBRN.2010.50.

[4] A. Menon, S. Singh, and H. Parekh, 'A Review of Stock Market Prediction Using Neural Networks', in *2019 IEEE International Conference on System, Computation, Automation and Networking (ICSCAN)*, Pondicherry, India, Mar. 2019, pp. 1–6, doi: 10.1109/ICSCAN.2019.8878682.

[5] J. Brownlee, 'How to Choose a Feature Selection Method For Machine Learning', *Machine Learning Mastery*, Nov. 26, 2019. https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/ (accessed Jan. 26, 2020).

[6] D. Soni, 'Introduction to Evolutionary Algorithms', *Medium*, Jul. 16, 2019. https://towardsdatascience.com/introduction-to-evolutionary-algorithms-a8594b484ac (accessed Jan. 21, 2020).

[7] S. Patnaik, Ed., *Recent Developments in Intelligent Nature-Inspired Computing:* IGI Global, 2017.

[8] DataVedas, 'EMBEDDED METHODS | Data Vedas'. https://www.datavedas.com/embedded-methods/ (accessed Jan. 28, 2020).

[9] T. Lin, T. Guo, and K. Aberer, 'Hybrid Neural Networks for Learning the Trend in Time Series', in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, Melbourne, Australia, Aug. 2017, pp. 2273–2279, doi: 10.24963/ijcai.2017/316.

[10] '(1) (PDF) An Adaptive Offloading Method for an IoT-Cloud Converged Virtual Machine System Using a Hybrid Deep Neural Network', *ResearchGate*. https://www.researchgate.net/publication/328636008_An_Adaptive_Offloading_Method_for_an_IoT-Cloud_Converged_Virtual_Machine_System_Using_a_Hybrid_Deep_Neural_Network (accessed Mar. 01, 2020).

[11] J. Eapen, D. Bein, and A. Verma, 'Novel Deep Learning Model with CNN and Bi-Directional LSTM for Improved Stock Market Index Prediction', in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, USA, Jan. 2019, pp. 0264–0270, doi: 10.1109/CCWC.2019.8666592.

[12] C. Liu, K. Li, J. Liu, and C. Chen, 'LHCnn: A Novel Efficient Multivariate Time Series Prediction Framework Utilizing Convolutional Neural Networks', in *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, Zhangjiajie, China, Aug. 2019, pp. 2324–2332, doi: 10.1109/HPCC/SmartCity/DSS.2019.00323.

[13] 'images (Imagen PNG, 270 × 187 píxeles)'. https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcS6o-bFFynY9S_urprOHPLGPf_MWzRWR8dTohZujIhl9gdRKuKm&usqp=CAU (accessed Apr. 23, 2020).

[14] A. Bissuel, 'Hyper-parameter optimization algorithms: a short review', *Medium*, Apr. 24, 2019. https://medium.com/criteo-labs/hyper-parameter-optimization-algorithms-2fe447525903 (accessed Feb. 05, 2020).

[15] J. Brownlee, 'Understand the Impact of Learning Rate on Neural Network Performance', *Machine Learning Mastery*, Jan. 24, 2019. https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/ (accessed Feb. 04, 2020).

[16] 'machine learning - Epoch vs Iteration when training neural networks', *Stack Overflow*. https://stackoverflow.com/questions/4752626/epoch-vs-iteration-when-training-neural-networks (accessed Feb. 04, 2020).

[17] 'Hyperparameters: Optimization Methods and Real World Model Management', *MissingLink.ai*. https://missinglink.ai/guides/neural-network-concepts/hyperparameters-optimization-methods-and-real-world-model-management/ (accessed Feb. 03, 2020).

[18] 'Hyperparameter optimization for Neural Networks — NeuPy'. http://neupy.com/2016/12/17/hyperparameter_optimization_for_neural_networks.html#hand-tuning (accessed Feb. 05, 2020).

[19] 'The Unreasonable Effectiveness of Quasirandom Sequences | Extreme Learning'. http://extremelearning.com.au/unreasonable-effectiveness-of-quasirandom-sequences/ (accessed Feb. 05, 2020).

[20] A. Osipenko, 'Genetic algorithms and hyperparameters — Weekend of a Data Scientist', *Medium*, May 30, 2019.

https://medium.com/cindicator/genetic-algorithms-and-hyperparameters-weekend-of-a-data-scientist-8f069669015e (accessed Feb. 06, 2020).

[21] A. Li *et al.*, 'A Generalized Framework for Population Based Training', in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining - KDD '19*, Anchorage, AK, USA, 2019, pp. 1791–1799, doi: 10.1145/3292500.3330649.