algorithms

MDPI

Article

Order-Based Schedule of Dynamic Topology for Recurrent Neural Network

Diego Sanchez Narvaez, Carlos Villaseñor, Carlos Lopez-Franco and Nancy Arana-Daniel *

Department of Computer Science, University of Guadalajara, 1421 Marcelino Garcia Barragan, Guadalajara 44430, Jalisco, Mexico; diego.sanchez8805@alumnos.udg.mx (D.S.N.); carlos.villasenor@academicos.udg.mx (C.V.); carlos.lfranco@academicos.udg.mx (C.L.-F.)
* Correspondence: nancy.arana@academicos.udg.mx

Abstract: It is well-known that part of the neural networks capacity is determined by their topology and the employed training process. How a neural network should be designed and how it should be updated every time that new data is acquired, is an issue that remains open since it its usually limited to a process of trial and error, based mainly on the experience of the designer. To address this issue, an algorithm that provides plasticity to recurrent neural networks (RNN) applied to time series forecasting is proposed. A decision-making grow and prune paradigm is created, based on the calculation of the data's order, indicating in which situations during the re-training process (when new data is received), should the network increase or decrease its connections, giving as a result a dynamic architecture that can facilitate the design and implementation of the network, as well as improve its behavior. The proposed algorithm was tested with some time series of the M4 forecasting competition, using Long-Short Term Memory (LSTM) models. Better results were obtained for most of the tests, with new models both larger and smaller than their static versions, showing an average improvement of up to 18%.

Keywords: grow and prune paradigm; Cao's order analysis; recurrent neural networks; dynamic topology

Citation: Sanchez Narvaez, D.; Villaseñor, C.; Lopez-Franco, C.; Arana-Daniel, N. Order-Based Schedule of Dynamic Topology for Recurrent Neural Network. *Algorithms* **2023**, *16*, 231. https://doi.org/10.3390/a16050231

Academic Editor: Grazziela Patrocinio Figueredo

Received: 30 March 2023
Revised: 24 April 2023
Accepted: 26 April 2023
Published: 28 April 2023

1. Introduction

Recurrent Neural Networks (RNN) have proved to be a powerful methodology to solve time series forecasting [1,2], due to their ability to adapt and recognize short-term and long-term correlations between input data sequences [3].

The capacity that these systems have, that is, how much they can learn and store knowledge, is determined among other features by their topology [4–6], made up of the number of units that they have and how these units are interconnected, which allows the system to acquire, store, and generalize knowledge. In addition to its topology, the capacity of the network is dependent on the training process that is used. Faced with this situation, the question arises about how the network should be designed, a situation that even today has not been resolved, can be provided some theoretical guidelines [7] that suggest what is likely to work or not work on a given problem, but, ultimately, it has been limited to a process of trial and error, based only on the experience of the designer.

Depending on these two factors: topology, and training, it is possible that the short-term memory effect occurs, which causes that, as the network processes more future elements of the series, the parameters that represent the knowledge of the behavior of the sequence in the long term can vanish [8]. This makes it necessary to update the network by adding the new observations acquired over time, a process normally addressed by discarding the previous model and retraining the network from scratch using the new data set. One neural model that successfully deals with vanishing gradients of long-term temporal sequences is the Long-Short Term Memory model (LSTM) [9]. LSTM model adds

a constant error carrousel unit (CEC) that allows keeping gradients of data sequences for a long time.

Since time series sequences could suddenly change their complexity from one sample time to another, more or less neural units of the model can be needed to process dynamic data, and the LSTM efficiency and precision can be improved by adding functionalities inspired by concepts of biological brain neuroplasticity in the design and updating of architectures during the training time [10]. Some algorithms have been designed to simulate how synaptic connections are modified in biological neuronal networks and therefore how the number of neurons involved in a network is affected, developing techniques such as pruning [11] or growing [12], which represent brain plasticity. Brain plasticity describes the ability of brain areas or neuronal groups to respond functionally and neurologically to their environment, changing their activity in response to intrinsic or extrinsic stimuli by reorganizing their structures, giving them the possibility of growing and/or pruning synapses to fill functional deficiencies [13]. This is why, the objective of these techniques, growing and pruning, is to adapt the architecture of the networks that are already designed, increasing, or decreasing the number of neurons respectively, to deliver an efficient model that improves its performance. In this work the growing and pruning paradigm is developed using LSTM architectures to obtain LSTM models with better performance to solve time series forecasting problems.

Based on these methodologies, is proposed an algorithm that provides plasticity to an LSTM model. Which gives it the capacity to reduce or increase the number of neurons of its architecture in order to increase its precision and/or efficiency, applied to time series forecasting. This algorithm is a decision-making grow and prune paradigm, based on the calculation of the order of the data, being this, the way to simulate the stimulus presented in the environment, denoting to the network in which situations it should increase or reduce its connections, obtaining, as a result, a dynamic architecture that can facilitate its design and implementation, as well as improve its time series forecasting, obtaining reduced or larger models, but in both cases improving its performance. The algorithm consists of an automatized module that computes the order of the input data sequence, in this way the system will be able to detect the changes that the data presents. This calculation is carried out using Cao's methodology [14], which is an improvement of the False Nearest Neighbor, (FNN) [15], for determining the minimum embedding dimension. After Cao's order computation, a set of error metrics and a history of actions previously taken by the system will be fed into the decision-making policy to determine the next change that should be made in the LSTM topology. These changes can be apply: "pruning" eliminating connections of the network based on its magnitudes, "growing" activating connections, or increasing the number of neurons concerning the gradients or maintaining the network with its current structure.

Currently, the algorithm is designed to be applied when new information is obtained and the model needs to be retrained, a process that is called phase. At each phase, the decision policies will be activated and will indicate the modifications to be implemented in the topology during this process (training). This proposed methodology was tested using some time series obtained from the 2019 M4 contest daily training dataset [16]. A comparative analysis was performed to show the advantages of the methodology: the results obtained using a standard LSTM model were contrasted against those obtained using the dynamic architecture version of the LSTM.

The main contribution of this article is the proposal of a novel order-based schedule of dynamic topology for LSTM models, based on Cao's order analysis of time series and the evolution of the forecasting error.

Where in most of the tests, improvements were presented by reducing the final error obtained, and in some cases, they also reduced the original architecture of the network. In the tests that did not present considerable improvements, a similarity in the error range to its static counterpart was maintained, but the number of connections was decreased.

Related Work

Since the design of neural networks is a dilemma in the field of artificial intelligence, some models that implement these growth and/or pruning methodologies have been proposed to address this problem.

However, using growing and pruning methodologies in RNN is difficult since these networks deal with time-varying data. Due to the above, common metrics that are used to decide to grow/prune network architectures that process static data are no longer applicable. For example: if the decision to prune the network is taken based on low values of gradients of some neurons, this could happen because those neurons are essential only to recognize key correlations between long-term data correlations, and that's why they are just firing on the time that these correlations are presented in the data, which could not be frequent. On the other hand, a temporal decrease in the error value could indicate that the current architecture of the network is being enough to process the current data sequence. Nevertheless, the complexity of the time-varying data could change, and the network could require a sudden and critical increase of neurons which can be very computationally expensive.

So, there are a few approaches that present dynamic architecture RNN: among them is "Nonlinear Identification Via Variable Structure Recurrent Neural Networks" [17], which shows an algorithm to perform nonlinear identification via variable structure RNN; here, the growth technique is applied, increasing the size of the network by introducing high-order connections, limiting itself to the minimum required to improve its performance during the analysis of the data available, but at the same time, it does not take into account the possibility of decreasing the number of connections to obtain compact networks, thus increasing the probability that the topology has redundant connections. Another algorithm that applies these methodologies seeking to obtain a dynamic topology is the model "Dynamic Generation of the Topology of an Artificial Neural Network of the Multilayer Perceptron Type" [18], this addresses the situation by creating a dynamic tool that allows the user to design and modify a multilayer network by adding or removing synaptic connections using both pruning and growing techniques. Seeking to reduce the error by increasing or decreasing the connection nodes, but maintaining the decision only on the user and his design ability. "A Dynamic Ensemble Learning Algorithm for Neural Network" [19], is a model designed for classification problems, which looks to modify the topology starting from a multilayer neural network that will add more multilayer subnetworks that are designed in parallel, a process in which are added or deleted connections of the new subnet to be annexed. Generating these subnetworks automatically during the analysis process of the system eliminates user dependency, but with the particularity that the minimum possible model is a network with the addition of a sub-network, so the main model cannot be modified.

The model "Incremental Learning Using a Grow-and-Prune Paradigm with Efficient Neural Networks" [20] presents both methodologies, by proposing a consecutive growth paradigm followed by a pruning process that restores and eliminates redundant model connections. This algorithm was applied in convolutional networks, starting with a sparse network, and then applying the paradigm in its training process, but not taking into account the option to increase the number of units of the original model.

2. Theoretical Frameworks

One of the bases of the proposed algorithm is the calculation of the order of the data, thus seeking to obtain information on the inherent dynamics of the system. This calculation is carried out employing the methodology "Practical Method for Determining the Minimal Embedding Dimension of a Scalar Time Series" [14], which estimates the minimum embedded dimension based on the "False Nearest Neighbors" (FNN) [15] algorithm but looks for a more practical and simple way to obtain this metric.

2.1. Practical Method for Determining the Minimal Embedding Dimension of a Scalar Time Series

This method, proposed by Liangyue Cao [14], eliminates the need to make use of the threshold parameter used by the FNN algorithm ($Rtol$), which determines if the neighbor of a point is false or not, by describing it as a subjective parameter.

Let $X = x_1, x_2, x_3, ..., x_N$ be a time series, and y a d-dimensional vector defined as.

$$y_i(d) = \left[x_i, x_{i+\tau}, \ldots, x_{i+(d-1)\tau}\right] i = 1, 2, \ldots, N - (d-1)\tau \tag{1}$$

where both i and τ are values indicating the index of the original series data, where τ is the shift in the data and d is the dimension under analyzed. Similar to the FNN algorithm is defined:

$$a(i,d) = \frac{\left|\left|y_i(d+1) - y_{r(i,d)}(d+1)\right|\right|}{\left|\left|y_i(d) - y_{r(i,d)}(d)\right|\right|} i = 1, 2, \ldots, N - d\tau \tag{2}$$

Being the relationship between the distance of a point and its nearest neighbor in a larger dimension $(d+1)$ and the distance between the same points (anchor point and its neighbor) but in the dimension of analysis (d). If d qualifies as an embedded dimension, then any pair of close points in d must remain close in $d+1$, if this is true these two points will be described as true neighbors, otherwise, they will be false neighbors. This is the base of the FNN algorithm. From this result, FNN determines the status of the neighbors by comparing the change ratio between both points with respect to a threshold $(Rtol)$, so Cao substituted this threshold by implementing the following equation:

$$E(d) = \frac{1}{N - d\tau} \sum_{i=1}^{N-d\tau} a(i,d) \tag{3}$$

This is the average of all distance ratios $(a(i,d))$. Where $E(d)$ only depends on d and τ. In turn, $E1(d)$ is defined, which shows the variation of the averages during the expansions that exist from d to $d+1$.

$$E_1(d) = \frac{E(d+1)}{E(d)} \tag{4}$$

where instead of using the threshold proposed in FNN, this method observes $E1(d)$, which will stop changing when d is greater than a correct embed dimension (dx) so the minimum dimension will be $(dx+1)$.

2.2. Long Short-Term Memory (LSTM)

LSTM units are an extension of recurrent networks that allow them to remember their inputs for a longer period, resembling a computer's memory. It can read, write, and delete information from its cell, saving information, and preventing it to vanish through time [21].

The cell decides whether to store or delete the information it is receiving based on a degree of importance that is assigned to it. This importance is assigned through the weights of the connections, so the LSTM learns over time to identify what part of the information is important and what is not. Implementing an internal state of the cell that will generate a long-term memory [22].

An LSTM neuron is composed of three gates: input, forget, and output, as shown in Figure 1. Which determines if a new input is allowed, which information is eliminated, or if it is allowed to affect the output at the current time.

The first gate that appears from left to right in Figure 1 is the forget gate, which decides what information of the cell state (C^t) is going to be discarded, through a sigmoid function (σ).

$$f_t = \sigma\left(W_f * [h_{t1}, x_t] + b_f\right) \tag{5}$$

where (h_{t-1}) is the previous output of the unit, (x_t) is the input, (W_f) the forget gate weight matrix and (b_f) the gate bias. The output will be between a value ranged from 0 to 1, with 0

representing total elimination and 1 keeping the whole information. After this step, the input gate continues, which indicates what portion of the new information will be stored in the cell state. This gate is composed of two functions, one sigmoidal and the other hyperbolic tangent, which determines the degree of update of the state.

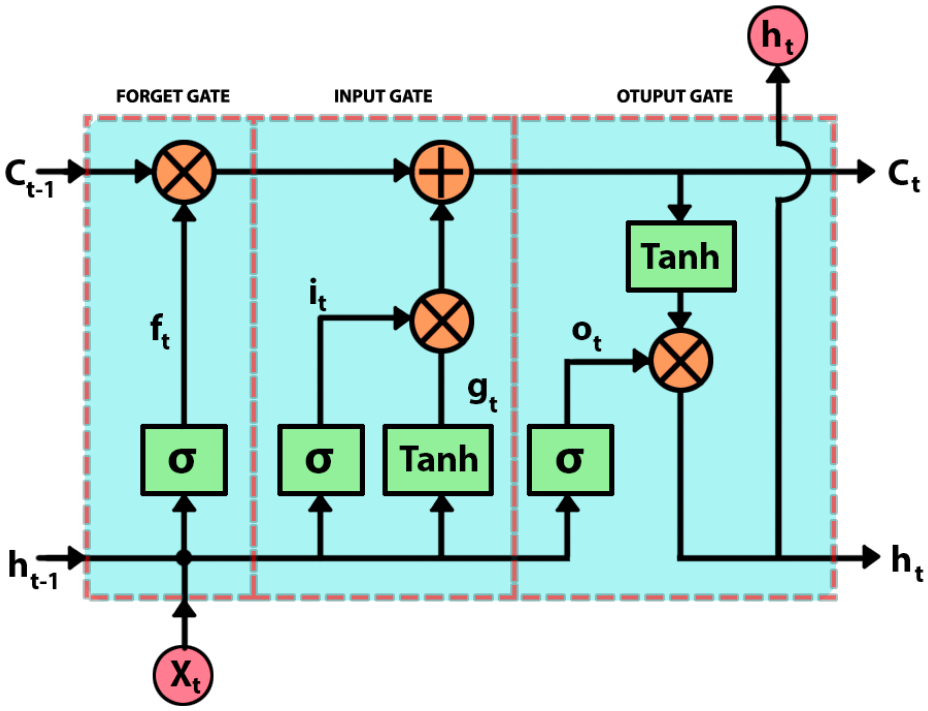

Figure 1. LSTM neuron structure (Forget gate, Input gate, Output gate) σ stands for sigmoid function, X_t the input data, and h_t the current network output.

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i) \tag{6}$$

$$g_t = \tanh(W_g * [h_{t-1}, x_t] + b_g) \tag{7}$$

Being (W_i) and (W_g) the weight matrix for the input gate and (b_i) and (b_g) their respective bias. Thus obtained, the new state of the unit, by eliminating the unnecessary information and updating it with the new information acquired.

$$C_t = [C_{t-1} * f_t] + [i_t * g_t] \tag{8}$$

Finally determine the output of the unit (h_t), which will be a version of the cell state filtered by the current input and the previous output. Where W_o and b_o are the weight matrix and bias for the output gate.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{9}$$

$$h_t = o_t * \tanh(C_t) \tag{10}$$

3. Dynamic Topology Method for RNN Based on Cao's Order

This section describes the processes that make up the proposed dynamic structure algorithm for recurrent networks, segmented into five modules:

- Order analysis
- Growing
- Pruning
- Normal

- Decision-Making policy

As it was mentioned previously, the algorithm is based on the elaboration of a decision-making policy that allows determining the action to be carried out on the LSTM network architecture (growing, pruning, or normal) with respect to the evolution of the data during the update process, in such a way that it allows the network to dynamically adapt its architecture to maintain, transfer and distribute the knowledge to its next version. The flowchart of the methodology is illustrated in Figure 2.

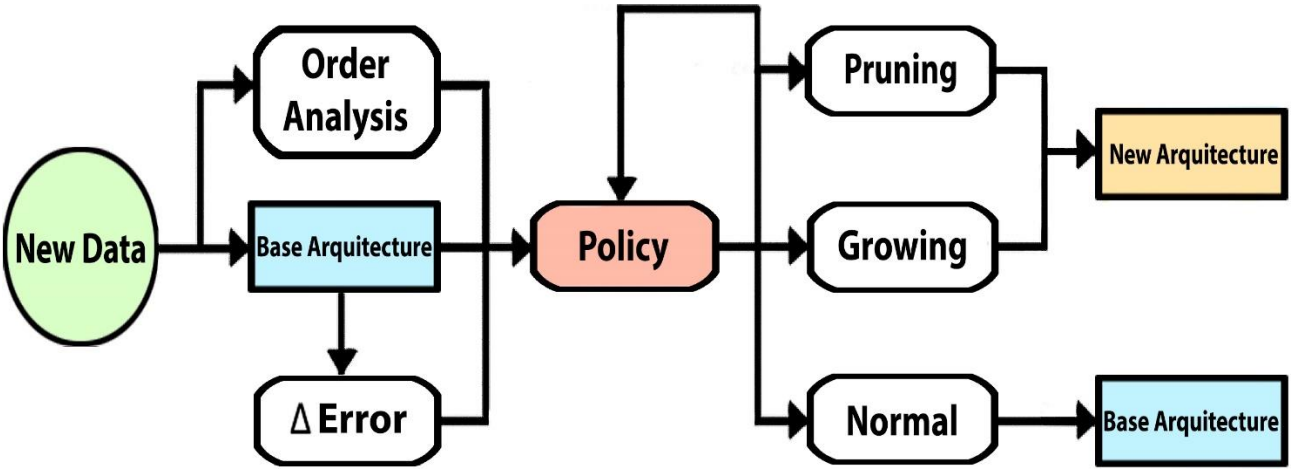

Figure 2. Dynamic architecture algorithm for RNN flowchart.

Starting from the current architecture and weights of the network, the process of evolution of the topology continues. If the network is created from scratch, it will be initialized randomly creating a base model. Once is received a base model, the methodology analyzes the new training data determining the order of this sequence employing "Cao's" algorithm to be introduced with other metrics in the decision policy that will select whether the network's architecture will get into: (1) a growing process, creating new connections or new neurons based on the information of its gradients; (2) into a pruning process, deleting redundant connections based on the magnitudes of its connections; or (3) preserving its current topology.

These processes, the metrics used, and the decision-making policy are described in the following sections.

3.1. Order Analysis

When a sequence of new data is acquired, this is preprocessed by an algorithm that will calculate its order. The aim of analyzing the order of the series is to have a metric that allows visualizing the evolution that the data is presenting and thus, in parallel, to modify the structure of the network so that it can resemble approximately the same dynamics.

To calculate the order, the algorithm Practical Method for Determining the Minimum Embedding Dimension of Scalar Time Series proposed by Cao, described in Section 2, is used. This is an improvement to the algorithm False Nearest Neighbors.

3.2. Growing Phase

Once the new data set has been collected and analyzed, through the evolution policy, it is determined if the network should: (1) Grow its architecture, (2) Keep its current architecture or (3) Prune the network by deleting some connections.

To expand its topology, an increase in the network architecture is carried out by adding neurons or, in other cases, activating connections using a gradient-based growth algorithm. The methodology makes use of binary masks that distinguish the dropped connections in each of the weight tensors of their respective layers. Identifying with "0" the connections that are turned off and with "1" the active ones. These masks, in turn, have the objective of

inhibiting the updating of non-active weights during the training process. This is divided into two steps, starting only with the new data set where the network growth will proceed and ending with a tuning training using the total data set (the previous data and the new ones collected) adapting the architecture to organize the new information available.

The network can present two possible scenarios when the decision to increase the topology has been taken, (1) the first occurs when the architecture has dropped connections, in this situation a percentage of the inhibited connections is reactivated. (2) If the network is fully connected, then, the growth of the network is carried out by adding neurons to the architecture.

3.2.1. Network Architecture Growth through Activation of Dropped Connections

The system allows the models to store their respective masks, so it can be identified if the network has non-activated connections. If this occurs, a sequential process is used to activate them. The growth of the network is determined by a rate parameter that tells the algorithm the percentage of the total units that must be activated in the network. In the experiments was used a 15% rate, which is applied every time that this process is made. This rate was chosen through experimentation.

Gradient evaluation: It starts with the calculation of the gradients of the dropped connections. These are calculated during training, where the gradients of all the weights are extracted during the backpropagation process, accumulating the gradients of only one epoch. It should be noted that the weights are not updated during this epoch. Then, the average of the resulting gradients is calculated.

Connection growth: The difference between the number of current connections and the percentage that must be initiated is calculated, obtaining as a result the number of dropped connections to be reactivated. This process is carried out for each set of weights of their respective layer. The connections with the highest gradients will be identified and then activated by changing their current value from 0 to 1 in their mask, that is, only the connections with the highest gradient will be activated until the percentage of active connections is completed. Seeking to have the most efficient connections that reduce the cost function.

Initialization: The new connections are initialized with the value resulting from the multiplication of the current training rate and their respective gradients, calculated in the previous step.

Once the new connections have been initialized, the updated model is trained using their masks, thus preventing the reactivation of the remaining dropped connections.

3.2.2. Network Architecture Growth by Adding New Neurons

If the model has all its connections activated but the method is indicating that the network must grow, a process of annexing new units is carried out to obtain a topology greater than the current one. Having the model, the option of increasing the number of the original model units. As in the previous situation, a sequential process is used starting with the calculation of gradients.

Gradient evaluation: Unlike what was previously explained, where the gradients were calculated and averaged only for the dropped connections since in this case they are all activated, the gradients are calculated and averaged, but respect to each layer, in order to identify which layer is the one that presents a greater gradient and try to reduce the cost function by introducing new units to the layer with the highest gradient.

Network expansion: Once the layer to be increased is determined, an expansion procedure must be carried out where one unit is added to the selected layer, expanding the connections of the neighbor units in the same and the next layer. To generate these connections, tensors are concatenated to the weight matrices in such a way that the missing connections are assembled. This procedure maintains the current values of the layers, thus avoiding the loss of knowledge of the model and being able to transfer it to its new version.

Initialization: To initialize the new tensors, a range of values is determined among which a value will be randomly assigned to each element of the tensor. This range is calculated with reference to the maximum and minimum value of each weight matrix, so each initialization will be dependent on the weight matrix. In this way, the aim is to prevent the new weights from presenting a bias away from the current search area, and having abrupt changes in the gradient values during training sessions.

In the end, the network is trained to adjust the values to the new data acquisition.

3.2.3. Growing Process

The growing phase is an iterative process. Divided into 3 steps.

In the beginning, the model received is trained for a small fixed number of epochs, this is made in order to obtain a metric that helps to compare the performance of the model and be able to distinguish if the growing process is improving the model. In the experiments, the metric used was the "root mean squared error" of the validation data during the training process, saving the best value reached.

Once this first step is done, the model grows using one of the two options explained before (Activation of dropped connections or Growth of neurons) using only the new data set and is trained during a fixed number of epochs. After this, the model is pruned 15% of its connections, letting the model grows in the next step, but now with the whole training set (the previous data and the new ones collected). Being this the second step.

The third step as it was mentioned before, reactive the 15% of the model that was pruned but now use the whole training set. Next, a tuning training is performed. During this, the metric root mean squared error is compared to the one saved in the first step, if this metric reaches a better result the training stops, the new best result and the weights are saved, and steps 2 and 3 are made again until the metric can't reach a better result. At this moment, if the model didn't grow a new neuron during the last step 2, the model will be set with the last weights saved, but if the model added a new neuron, the model will stay with the last set of weights trained during step 3.

This process gives the model the ability to grow iteratively as long as a better result can be achieved or until the model adds neurons three times.

3.3. Pruning Phase

Models presenting a large number of parameters are usually redundant [23], so there are methodologies to reduce this redundancy. In the proposed algorithm, a pruning policy is applied concerning the magnitudes of the weights. Where the weights with the smallest magnitudes will be inhibited, making an analogy to the disconnection of neural synapses.

As in the reactivation of connections, the same rate value is used to calculate the percentage of the network that will be pruned, in a similar way to the process described in the growth of connections, the number of connections to "turn off" will be the difference between the current deactivated connections and the ones remaining to reach the desired rate. The weights of the connections to be disabled are modified iteratively, assigning them to 0. Once the number of disconnections is reached, their new mask is calculated, updating the new disabled connections to 0.

Pruning Process

As in the growing process, the pruning phase is iterative. It's made of 2 steps, being the first step the same as the one explained in the growing process, where the received model is trained and saves the best root mean squared error value of the validation set during this training. It's necessary to mention that in both steps the whole training set is used, instead of only the new data acquired.

For the second step, the 15% of the model is pruned and then a tuning training is made, where the root mean squared errors are compared. If a better result is achieved the training is stopped, and the new best value and the weights are saved. This step is repeated until the model can't reach a better value. When this happens the model is set with the last

saved weight, which means, the model will reactive the 15% of the model before the 15% that couldn't get a better result, this happens only if step 2 was made at least one time. If not, the model will be pruned and will keep the tuning weights.

The model can be pruned until 80% of the model is pruned, once this rate is reached, if the model keeps getting better results, the model will continue training but without pruning it.

3.4. Normal Phase

When the algorithm indicates that is not necessary to change the topology of the model a "normal phase" is run. In this phase, the model keeps its architecture, but the training process is changed. This phase can run two different options before training the model, in both processes just the whole training set is used.

The first one is used when the last action taken by the system was to prune the model. In this case, before the model is trained, a single 15% growing process is made, and after this, is made a single 15% pruning, in this way the system keeps the same topology, using a growing-pruning run. Once these two actions are done the model continues the training process.

The second option is applied when the last decision was growing or when the model has not presented any change (neither pruning nor growing). Like the first option, the topology is changed and recovered but this time a pruning-growing run is made, using the same rate as the first option (15%). In the end, the model is trained.

This change-recover policy helps the model to transfer the knowledge of the previous phase and reorganize the new information. This also avoids the model to be the same as a vanilla model if no variation under Cao's method is presented.

3.5. Decision-Making Policy

The proposed algorithm is based on the union of the pruning and growth methodologies, and the order calculated by Cao's method. To unify these methodologies, a system that indicates which one of the processes (growing, pruning, or normal) should be executed is proposed. Thus, implementing a decision-making policy that determines through some metrics the type of evolutionary process that the net should execute. The metrics used were: (1) the mean square error (MSE) at the time of model evaluation; (2) the value resulting from Cao's order calculation; and (3) the previous decision made by the system about the type of evolution process executed. These metrics are compared between the last training run and the one before it, which allows us to observe the trend that the metrics are presenting, whether the error in the forecasting processes has been increased or decreased, and how the data dynamic is presented.

Phase: Each time that a new sequence of data is presented to the network is named a phase of the system.

Previous Decision: This describes the type of evolution process that the algorithm executed on each phase and is the first element taken into account to make the next decision about the type of network evolution that the system will execute.

Error: Continuing with the second element of comparison, it is the error calculated during the prediction of each phase. It is worth mentioning that the prediction is made concerning and during the acquisition of the new data, calculating the MSE error of each new pattern. In addition to the calculation of the error, the results in phases t-2 and t-1 are compared and it is determined if the error increased or decreased (Δ Error).

Cao: The next element taken into account to make a decision is the metric called "Cao", which indicates the order presented by the new data sequence of the stage. Regarding the change presented between the new and last result, a suggestion of the following action that will be made is generated. If the order decreased compared to the previous one, pruning will be suggested, otherwise, if the order increases, growing will be requested.

Decision: Based on these metrics, a decision tree was made, which will evaluate element by element until it takes the final decision of what evolutionary process will be

executed for the new data sequence training. Figure 3 shows the tree diagram. If the last decision made was to maintain the network architecture without modifications and run a normal phase (N), the last decision made that is growing (G) or pruning (P) with its respective branch in the decision tree will be taken into account instead of N, identifying it in Figure 3 with the symbol (X).

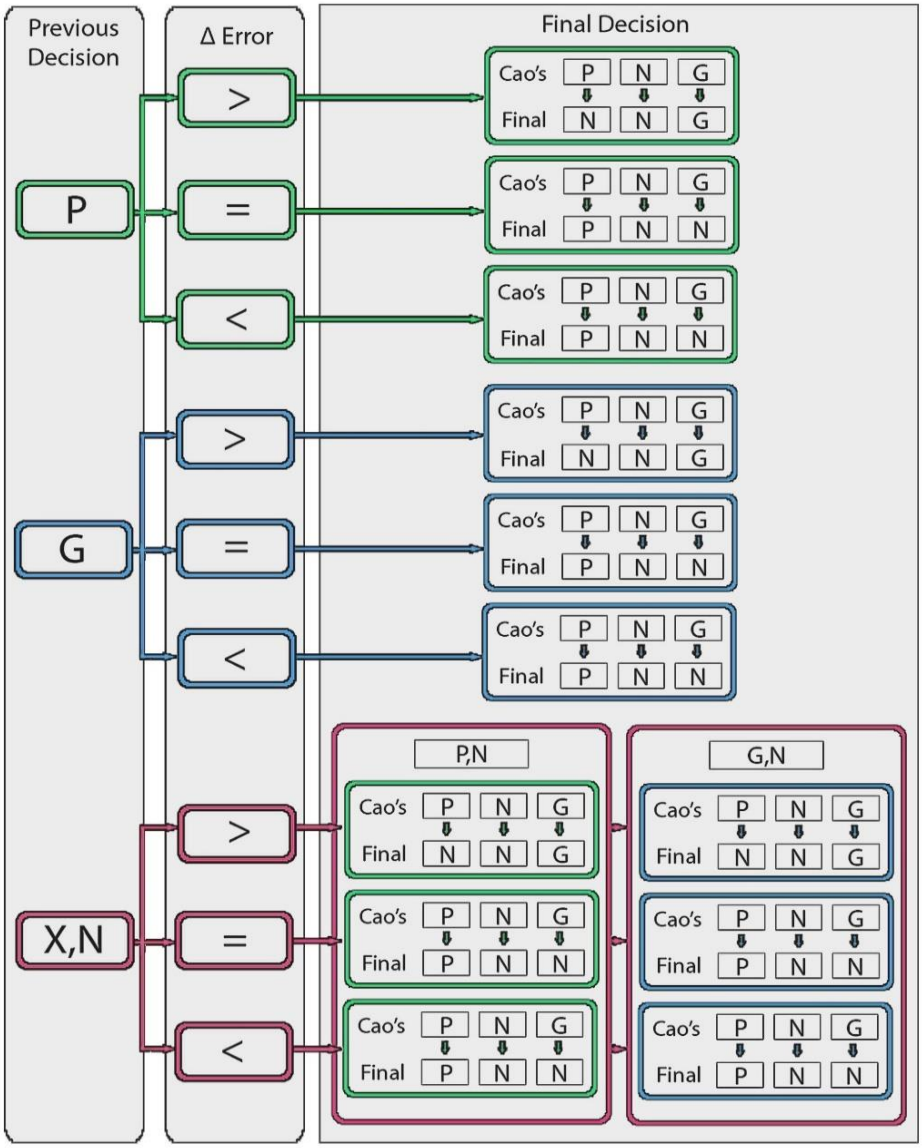

Figure 3. Decision tree, [P = Pruning, G = Growing, N = Normal, X = Last decision different from N.

The following table illustrates an example of the decision-making process regarding each training phase and the three metrics used.

Table 1 shows training phases starting at phase 0 with the standard model and current data. The symbols G, P, and N indicate the type of evolutionary process that was executed, i.e., if the network was grown, pruned, or if it maintained its topology, respectively. Meanwhile E_i stands for the value of the MSE of the $i-th$ phase. In this example are shown five phases, in Phase 0 the model was trained, and the error and Cao's metrics were calculated. The models only can change after Phase 0. In Phase 1 Cao's metric gave as result a bigger value than the one in Phase 0, because of this, the model suggested a growing run. The model ran Cao's suggestion. After this, the error is calculated during the test process and continues with the next phase.

Table 1. Decision-Making example by phases.

Phase	0	1	2	3	4
Previous Decision	-	-	G	P	P, N
Error	E_0	E_1	E_2	E_3	E_4
ΔError	-	-	$E_0 > E_1$	$E_1 > E_2$	$E_2 < E_3$
Cao	4	5	3	4	5
Cao's Suggest	-	G	P	G	G
Decision	-	G	P	N	G

Note: On each phase, the error is calculated, from phase 2, Δ Error is obtained by comparing the last two results, so can be determined if the error is being reduced or not.

From Phase 2 the algorithm starts using the 3 metrics. In this phase the previous decision made was growing (Previous Decision row), the error calculated in Phase 1 is bigger than the one calculated in Phase 0 (Δ Error row, Phase 2), and Cao's metric calculated in this phase is smaller than the phase before (Cao row, phase 1, Phase 2), which made the pruning suggest as can be seen in Cao's suggest Phase 2. As a result of all this information, the algorithm applied a pruning run.

Phase 3 presented a bigger order than Phase 2 and even though Cao's suggestion was, to apply a growing run, the algorithm didn't apply it, instead, it ran a normal stage. Because the policy considers the 3 metrics used (Previous Decision, Δ Error, and Cao's suggest) and not only Cao's suggest as can be seen in Figure 3. All the decisions starting from phase 2 use the decision tree. The last phase of the example used the same logic.

4. Results

To validate the proposed system, a comparison was made between Classic LSTM models (often called Vanilla LSTMs) and LSTM networks implemented with the proposed methodology, which will be named DyLSTM models to identify them. The test process and the results obtained are described below.

Dataset: The process makes use of some time series obtained from the M4 forecasting contest daily training dataset. The time series used by M4 were selected from the database ForeDeCk [24], which contains time series built from multiple and diverse sources. These series came from domains such as industries, services, tourism, imports & exports, demographics, education, labor & wage, government, households, bonds, stocks, insurance, loans, transportation, and natural resources & environment. This dataset has several time series that have different numbers of observations, due to this, were selected randomly 10 series with 1000 observations and 4 series with 4400 observations. Each series was partitioned into 10 divisions of equal sizes. In each stage, there will be 2 subsets, the first for training and the second for the prediction process (test).

Starting the model with the first division to consequently feed the system with the following divisions. Creating the training subset of the current phase, by joining the two subsets of the previous phase. In this way, the scenario of the acquisition of new data is simulated. Figure 4 shows the distribution that the data will present for each test.

Training: In all the phases, the calculation of the order is carried out, as well as one of the three possible training processes is applied: "growing", adding new units and/or reactivating connections; "pruning", disabling connections; or "normal", training and maintaining the current network architecture.

For the selection of the training process, is used the decision-making tree, described in the previous section, except for the first two phases. The first phase is the initialization of the models, in which they are trained with the first subset without any modification, keeping their respective current architectures. The second stage will only take into account the difference in the result that Cao's methodology delivers in the current stage compared to the previous one, applying growing if the order change is greater, pruning if the order is decreased, or maintaining the architecture if the order did not change.

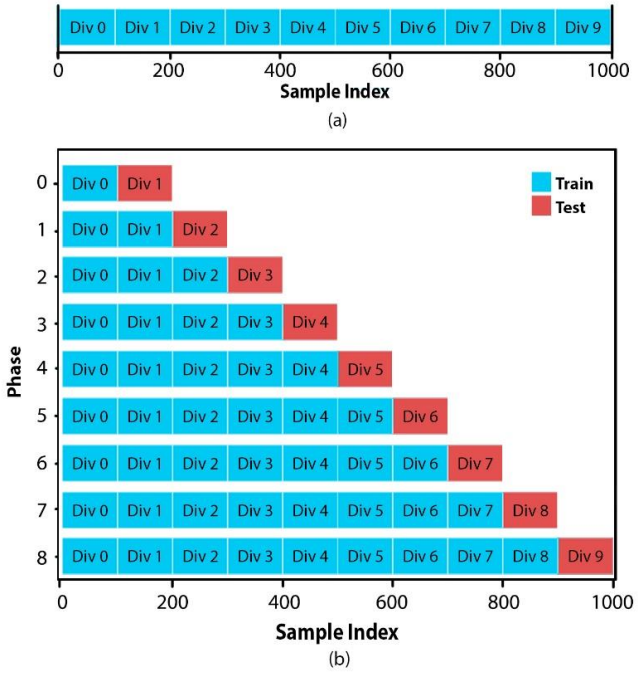

Figure 4. Data distribution: (**a**) Data set divisions, (**b**) Train and Test subsets on each phase.

Cao: Originally, the calculation of Cao was made to the division that was added to the training subset, this being the test subset in the previous stage $(Test[t-1])$, but looking toward increasing the capacity of the Cao algorithm and giving context to the data was decided to expand its range. Calculating Cao now using windows, being the window of the phase (U_t) the union of the divisions $(t-1)$ and (t), to later make the comparison with the previous window (U_{t-1}) composed by the divisions $(t-2)$ and $(t-1)$ and thus be able to determine if there is an increase or decrease in the order. It is worth mentioning that this calculation method has a variation in the first two stages, wherein in the first stage, the order of the first division is calculated and in the second stage the order of the union of the first two divisions is calculated and compared with the result of the previous stage omitting the division union $(t-2)$ as it does not exist. Figure 5 exemplifies the process of moving the windows during the phases.

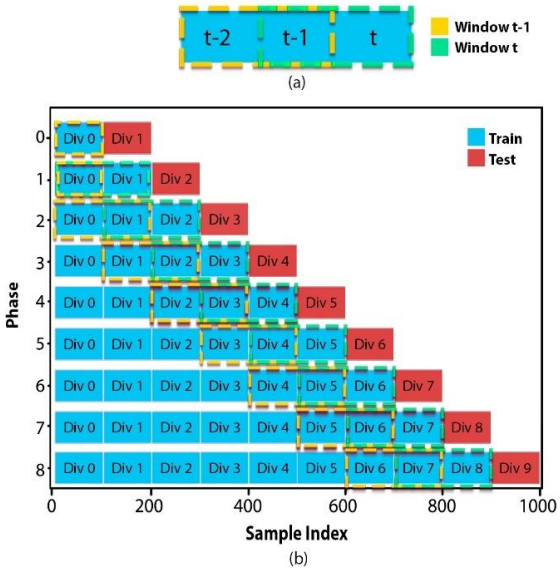

Figure 5. Mobile window for order calculation: (**a**) Generation of windows, (**b**) Movement of windows on each phase.

4.1. Test

Five recurrent models made up of two hidden LSTM layers and an output layer with only one neuron were created. Each model has a different number of LSTM units in the two hidden layers. Obtaining "many to one" LSTM models with one step horizon. These will be compared with their counterparts that will apply the methodology proposed, presenting two groups of models (LSTM Vanilla and DyLSTM), having ten models at the end, where the first group will be trained commonly and the second will be trained with the decision policies. The initial architecture of DyLSMT models will be clones of the Vanilla models, so both groups will be initialized with the same architectures and weights as their counterparts. Table 2 shows the five models used with their respective architectures.

Table 2. Models' architecture.

Model	Layer 1	Layer 2	Layer Out
M1	64	32	1
M2	32	16	1
M3	24	12	1
M4	18	6	1
M5	9	5	1

In each phase, the models of both groups will be trained concerning their type of training and the forecasting process will be carried out with the corresponding subsets of the test, gathering the resulting error for each input pattern, ending with the calculation of the average of all the errors of the phase in their respective model. To later compare both versions and detect if an improvement was made.

Performance metrics: There are many metrics in the literature for the evaluation of forecasting methods, so it was decided to use the metrics used by the M4 which is the Overall Weighted Average (OWA) metric, which computes an average of two evaluation metrics, being the mean square error (MSE) and the symmetric mean absolute percentage error (sMAPE), the ones used, to achieve a higher level of objectivity. The equations for each of the metrics are shown next.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{11}$$

$$\text{sMAPE} = \frac{1}{n}\sum_{i=1}^{n}\frac{\frac{|\hat{y}_i - y_i|}{|y_i|+|\hat{y}_i|}}{2} \tag{12}$$

where $\hat{y}$ is the forecast value and y is the actual value.

4.2. Experimental Results

To determine as objectively as possible whether the dynamic model presents an improvement with respect to the Vanilla model, the OWA metric is used, calculated from the MSE average and the sMAPE average of all the phases and not only of the last one, since, the objective of the proposed method is the reduction of the metrics values and in some cases, the reduction and improvement of the model, keeping its error as close or better to a standard version in each retraining process (when new data is acquired), helped by a knowledge transfer process, between each new architecture.

In each of the 14 series, the error metrics (MSE and sMAPE) were calculated for each phase and model, ending with the average of all the phases to determine the OWA result among the 5 models.

To calculate OWA, the rates between the error metrics of the two models to be compared are added, in this case, since it seeks to determine if there is an improvement in the dynamic model, the OWA is calculated as follows.

$$\text{OWA} = \frac{\text{Avg MSE}_{\text{MDi}}}{\text{Avg MSE}_{\text{Mi}}} + \frac{\text{Avg sMAPE}_{\text{MDi}}}{\text{Avg sMAPE}_{\text{Mi}}} \text{i} = 1, 2, 3, 4, 5 \quad (13)$$

where M is the vanilla version and MD is the proposed dynamic architecture version. If OWA is less than 1, it means that on average the model with the proposed algorithm has presented an improvement with respect to the Vanilla model. Where the difference to 1 is the average improvement percentage, that is, if the resulting value is 0.88, this would indicate that the model examined is 12% more accurate, taking MSE and sMAPE into account.

Figure 6 shows the evolution of the MSE error in each of the 10 models, the order obtained, and the decisions made at each phase during the first test (series 1). The regular lines describe the evolution of the errors obtained by the vanilla models and the dotted lines describe the errors acquired by the proposed models. The bars with the prefix TR show how the DyLSTM models were trained, and which change of architecture was run, being 0 as normal run, 1 as pruning run, and 2 as growing run being this metric measured in the right axes of the graph. The gray bar is the result of Cao's method, which is also measured in the right axes of the graph. Each model has its own color code: blue for the first models, red for the second ones, orange for the third, purple for the fourth, and green for the fifth models.

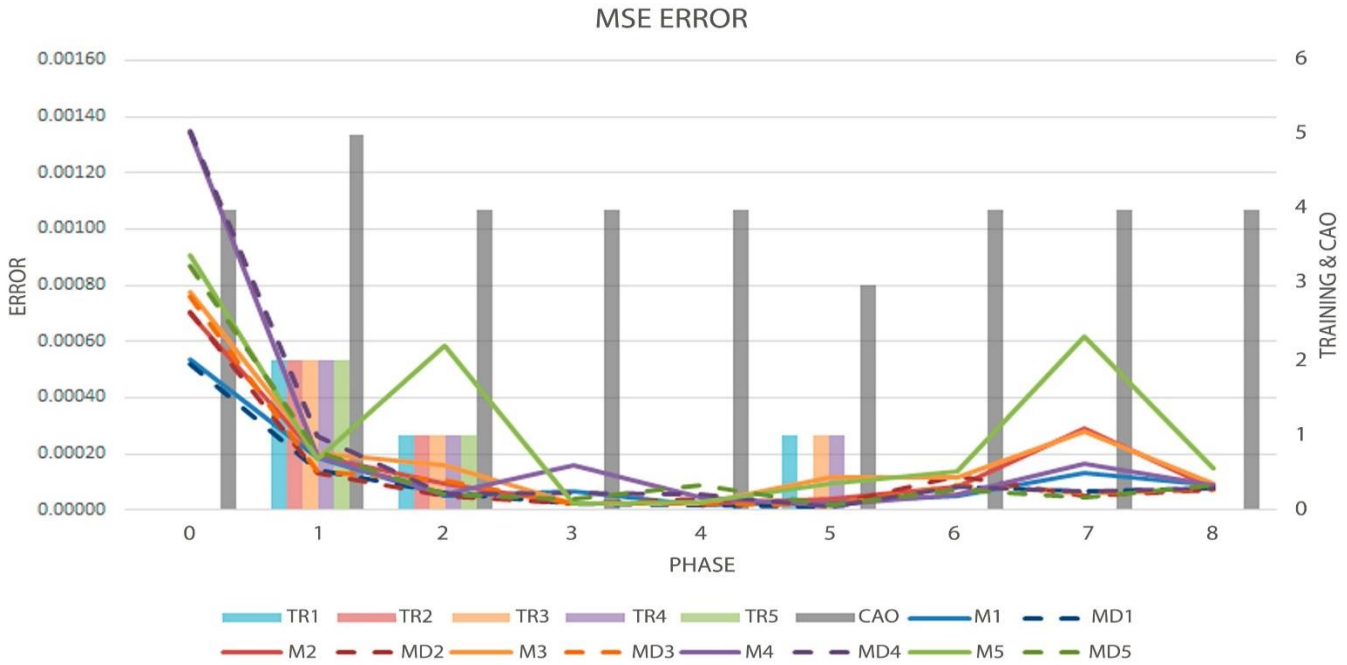

Figure 6. Graph of the MSE metric (Solid and Dotted Lines, measured by left axis), Order evolution (CAO), and the decisions made by the algorithm (TR-Bars, the last two metrics measured by the right axis), in the first test.

Figure 7 displays how the DyLSTM models change their topology and the final rate they achieved at each phase. The rate is determined by the difference with the original architecture, being this the architecture of the vanilla model. So, if a DyLSTM model has a rate under 1 means that at that point the model is smaller than the Vanilla, as the same if the model has a bigger rate, the DyLSTM model has added new units.

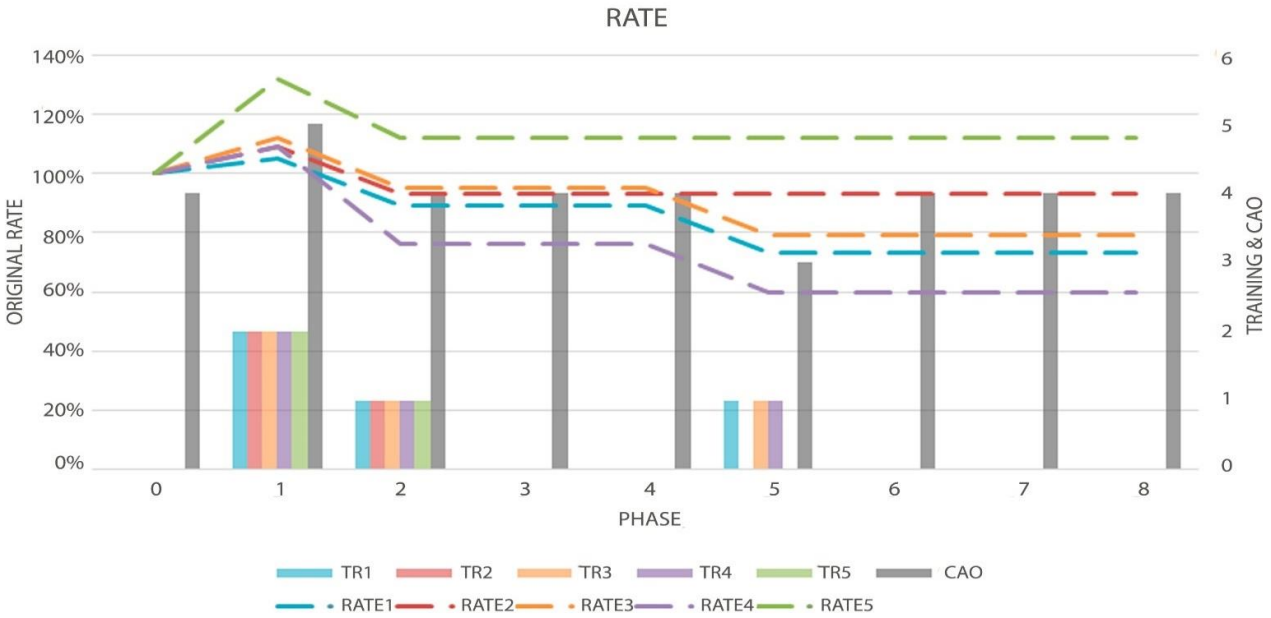

Figure 7. Graph of the Topology Rate (Dotted Lines, measured by left axis), Order evolution (CAO), and the decisions made by the algorithm (TR-Bars, the last two metrics measured by the right axis) in the first test.

In Figure 6 can be seen that in Phase 1 the five models' TR bars have a value of 2, which describes that the five models applied a growing run, and due to the models didn't apply a pruning run before, the models added units. This gave in result bigger models than the originals, as can be seen in Phase 1 Figure 7, where their rates are greater than 100%. In phase 2 the five models applied a pruning run, this is described by the five TR bars with a value of 1, and the results of the rate changes, made in the topology are presented in Figure 7 Phase 2. Tables 3 and 4 show the results of the MSE and sMAPE metrics of the first test, respectively. They show the model with its counterpart and the evolution of the metric on each phase, at the end, the average obtained from the complete test is shown.

Table 3. Series 1 (MSE), order [Cao] and comparison between the 5 models [M] and their respective dynamic versions [MD], rate, and the decision made [TR] (Growth = 2, Pruning = 1, Normal = 0).

MSE MODEL	**Phase 0**	**Phase 1**	**Phase 2**	**Phase 3**	**Phase 4**	**Phase 5**	**Phase 6**	**Phase 7**	**Phase 8**	**AVG**
M1	0.00053	0.00018	**0.00005**	0.00007	0.00002	0.00003	**0.00005**	0.00013	0.00009	0.00013
MD1	**0.00052**	**0.00015**	0.00006	**0.00002**	0.00002	**0.00001**	0.00008	**0.00007**	**0.00008**	**0.00011**
TR1	0	2	1	0	0	1	0	0	0	
RATE 1	1	1.05	0.89	0.89	0.89	0.73	0.73	0.73	0.73	
M2	0.00070	0.00019	0.00009	0.00002	**0.00002**	0.00004	**0.00008**	0.00029	0.00008	0.00017
MD2	0.00070	**0.00013**	**0.00005**	0.00002	0.00004	**0.00002**	0.00012	**0.00005**	**0.00007**	**0.00013**
TR2	0	2	1	0	0	0	0	0	0	
RATE 2	1	1.09	0.93	0.93	0.93	0.93	0.93	0.93	0.93	
M3	0.00078	0.00020	0.00016	0.00002	**0.00002**	0.00011	0.00011	0.00028	0.00010	0.00020
MD3	**0.00076**	**0.00014**	**0.00010**	0.00002	0.00003	**0.00002**	**0.00009**	**0.00006**	**0.00007**	**0.00014**
TR3	0	2	1	0	0	1	0	0	0	
RATE 3	1	1.12	0.95	0.95	0.95	0.79	0.79	0.79	0.79	

Table 3. *Cont.*

MSE MODEL	Phase 0	Phase 1	Phase 2	Phase 3	Phase 4	Phase 5	Phase 6	Phase 7	Phase 8	AVG
M4	**0.00134**	**0.00019**	0.00006	0.00016	0.00005	0.00002	**0.00005**	0.00017	0.00009	0.00024
MD4	0.00135	0.00026	0.00006	**0.00006**	0.00005	**0.00001**	0.00008	**0.00007**	**0.00008**	**0.00023**
TR4	0	2	1	0	0	1	0	0	0	
RTE 4	1	1.09	0.76	0.76	0.76	0.6	0.6	0.6	0.6	
M5	0.00091	**0.00018**	0.00059	**0.00002**	**0.00003**	0.00009	0.00014	0.00062	0.00015	0.00030
MD5	**0.00087**	0.00021	**0.00005**	0.00004	0.00009	**0.00002**	**0.00007**	**0.00005**	**0.00009**	**0.00016**
TR5	0	2	1	0	0	0	0	0	0	
RATE 5	1	1.32	1.12	1.12	1.12	1.12	1.12	1.12	1.12	
CAO	**4**	**5**	**4**	**4**	**4**	**3**	**4**	**4**	**4**	

Table 4. Series 1 (sMAPE), order [Cao], and comparison between the 5 models [M] and their respective dynamic versions [MD], rate, and the decision made [TR] (Growth = 2, Pruning = 1 Normal = 0).

sMAPE MODEL	Phase 0	Phase 1	Phase 2	Phase 3	Phase 4	Phase 5	Phase 6	Phase 7	Phase 8	AVG
M1	**37.97000**	6.12000	**2.13000**	2.18000	0.77000	0.84000	**0.78000**	1.14000	0.84000	5.86333
MD1	38.30000	**4.41000**	2.58000	**1.06000**	**0.69000**	**0.58000**	1.02000	**0.79000**	**0.78000**	**5.57889**
TR1	0	2	1	0	0	1	0	0	0	
RATE 1	1	1.05	0.89	0.89	0.89	0.73	0.73	0.73	0.73	
M2	44.21000	6.23000	3.34000	**1.04000**	**0.81000**	1.03000	**1.04000**	1.81000	0.78000	6.69889
MD2	**44.00000**	**3.78000**	**2.21000**	1.09000	1.17000	**0.62000**	1.30000	**0.70000**	**0.75000**	**6.18000**
TR2	0	2	1	0	0	0	0	0	0	
RATE 2	1	1.09	0.93	0.93	0.93	0.93	0.93	0.93	0.93	
M3	47.29000	6.79000	4.55000	**1.05000**	**0.81000**	1.91000	1.28000	1.72000	0.86000	7.36222
MD3	**47.04000**	**3.81000**	**3.49000**	1.07000	0.89000	**0.60000**	**1.07000**	**0.75000**	**0.76000**	**6.60889**
TR3	0	2	1	0	0	1	0	0	0	
RATE 3	1	1.12	0.95	0.95	0.95	0.79	0.79	0.79	0.79	
M4	**63.71000**	**6.28000**	**2.41000**	3.37000	**1.28000**	0.68000	**0.80000**	1.28000	0.83000	**8.96000**
MD4	63.80000	9.48000	2.48000	**1.99000**	1.36000	**0.58000**	1.03000	**0.77000**	**0.76000**	9.13889
TR4	0	2	1	0	0	1	0	0	0	
RATE 4	1	1.09	0.76	0.76	0.76	0.6	0.6	0.6	0.6	
M5	50.64000	**5.32000**	8.64000	**1.08000**	**1.00000**	1.69000	1.41000	2.73000	1.07000	8.17556
MD5	**50.63000**	7.11000	**2.06000**	1.50000	1.82000	**0.58000**	**0.97000**	**0.66000**	**0.81000**	**7.34889**
TR5	0	2	1	0	0	0	0	0	0	
RATE 5	1	1.32	1.12	1.12	1.12	1.12	1.12	1.12	1.12	
CAO	**4**	**5**	**4**	**4**	**4**	**3**	**4**	**4**	**4**	

Table 5 shows the results of the OWA calculation in each model, the final percentage of the dynamic architecture, with respect to the Vanilla version, as well as, the number of new units if they were added in the first test. Table 6 shows the result of Cao's method on each phase of the 14 series. Table 7 summarizes the results of the average of the 14 tests performed three times. As can be seen in Table 7, it is just displayed the final rate of the model but is necessary to remember that the topology of the models changes according to the variation of the order of the series and the decisions made by the policy. So, the last rate might not be the same in all the past phases.

Table 5. Metric (OWA). Final percentage (RATE) of the topology with respect to the initial model. (New units) Number of extra units added.

Model	OWA	RATE	New Units
M1/MD1	**0.91**	0.73	2
M2/MD2	**0.86**	0.93	2
M3/MD3	**0.81**	0.79	2
M4/MD4	**0.99**	0.6	1
M5/MD5	**0.72**	1.12	2

Table 6. Cao's method results from the 14 tests carried out.

	Phase 0	Phase 1	Phase 2	Phase 3	Phase 4	Phase 5	Phase 6	Phase 7	Phase 8
CAO S1	4	5	4	4	4	3	4	4	4
CAO S2	3	4	4	5	4	4	4	4	5
CAO S3	4	4	4	5	5	5	4	5	5
CAO S4	4	1	3	4	5	5	5	4	5
CAO S5	3	3	3	3	3	3	2	5	3
CAO S6	3	4	4	4	4	4	4	5	4
CAO S7	4	4	4	4	3	4	4	5	4
CAO S8	4	5	5	4	4	5	5	5	4
CAO S9	4	4	4	4	4	5	4	4	5
CAO S10	4	5	5	5	5	4	4	5	5
CAO S11	5	4	4	4	4	4	5	4	5
CAO S12	1	3	4	4	4	4	4	4	5
CAO S13	1	5	4	4	4	4	4	4	4
CAO S14	4	5	5	4	4	4	5	4	5

Table 7. Results of the 14 tests carried out.

MODEL	Serie 1			Serie 2			Serie 3			Serie 4			Serie 5		
	OWA	**RATE**	**UNITS +**	**OWA**	**RATE**	**UNITS +**	**OWA**	**RATE**	**UNITS +**	**OWA**	**RATE**	**UNITS +**	**OWA**	**RATE**	**UNITS +**
M1/MD1	**0.89**	0.783	2	**0.91**	0.91	3	**0.81**	1.05	2	**0.67**	1.07	3	1.03	0.86	1
M2/MD2	**0.87**	0.870	2	**0.79**	0.97	3	**0.93**	0.63	1	**0.83**	1.14	3	**0.66**	0.75	0
M3/MD3	**0.79**	0.790	2	**0.79**	0.99	3	**0.90**	0.69	1	**0.88**	1.27	4	**0.67**	0.76	1
M4/MD4	**0.76**	0.763	1	**0.61**	1.05	3	**0.88**	0.80	1	**0.97**	1.40	4	**0.59**	0.82	1
M5/MD5	**0.94**	0.937	2	**0.86**	1.23	3	1.07	0.98	1	1.03	1.78	4	**0.38**	0.79	1
MODEL	**Serie 6**			**Serie 7**			**Serie 8**			**Serie 9**			**Serie 10**		
	OWA	**RATE**	**UNITS +**	**OWA**	**RATE**	**UNITS +**	**OWA**	**RATE**	**UNITS +**	**OWA**	**RATE**	**UNITS +**	**OWA**	**RATE**	**UNITS +**
M1/MD1	**0.94**	1.05	2	**0.84**	0.50	0	**0.96**	0.87	1	**0.80**	0.87	1	**0.81**	0.88	2
M2/MD2	**0.90**	1.09	2	**0.88**	0.51	0	**0.96**	0.98	2	**0.76**	0.89	1	**0.89**	0.93	2
M3/MD3	**0.90**	1.12	2	**0.90**	0.74	1	**0.92**	0.89	2	**0.77**	0.90	1	**0.86**	0.95	2
M4/MD4	**0.92**	1.18	2	**0.88**	0.87	0	**0.80**	1.01	2	**0.85**	0.93	1	**0.94**	1.05	2
M5/MD5	**0.89**	1.31	2	**0.94**	0.74	0	**0.94**	1.10	2	**0.82**	0.97	1	**0.82**	1.38	2
MODEL	**Serie 11**			**Serie 12**			**Serie 13**			**Serie 14**					
	OWA	**RATE**	**UNITS +**	**OWA**	**RATE**	**UNITS +**	**OWA**	**RATE**	**UNITS +**	**OWA**	**RATE**	**UNITS +**			
M1/MD1	1.01	0.70	0	**0.60**	1.05	2	**0.67**	0.89	2	**0.99**	0.73	2			
M2/MD2	**0.75**	0.60	0	**0.70**	1.09	2	**0.72**	0.87	2	1.01	0.76	2			
M3/MD3	1.03	0.55	0	**0.58**	1.12	2	**0.60**	0.95	2	**0.87**	0.79	2			
M4/MD4	1.01	0.55	0	**0.51**	1.17	2	**0.49**	0.88	2	**0.94**	0.77	2			
M5/MD5	**0.66**	0.40	0	**0.56**	1.27	2	**0.60**	0.99	2	**0.91**	0.95	2			

Note: The table shows the OWA, the final percentage (Rate) of the dynamic topology with respect to the initial model, and if the addition of units in the model has been presented, the number of new units is displayed in the (Units +) section.

5. Discussion

Through experimentation, certain variations of the method of the proposed algorithm were designed, such as the extension of the range of the windows for the calculation of the order, given that, initially, when only the new data was analyzed, our algorithm determined to make abrupt changes to the network topology since the system had a lack of context of the previous data. When it was determined the need to expand the amount of data, adding the previous data window to the current one, an improvement in order detection was obtained.

Another section that presented modifications in the experimentation process was the generation of the decision-making rules of the tree as its hierarchy. Looking to create a dependency between the decisions made over time and the evolution of the error metrics, the previous decision was assigned in the first position of the decision-making hierarchy followed by the error variation, thus presenting a distribution of rules that decreased or maintained the trend of the error.

It is important to note that in almost all of the 14 tests, our approach improved the results obtained by the vanilla LSTM models; in the cases that didn't improve, the models keep almost the same performance as their counterparts. These cases were presented in five series (3, 4, 5, 11, 14) in just 7 models, where the worst model was presented in series 3, as can be seen in Table 7. During the experiments, in general, these results happened because in one of the phases, almost always after when a considerable change of order happened and a modification on the topology was made, the dynamic model had an error worse than the vanilla model, being this error enough to make on average that the dynamic model be worse in the whole test. As an example, Figure 8 and Tables 8 and 9 is presented one of the tests of series 11, which had a major number of models with no improvement, where can be seen that models 1 and 4 had an error during phase 2 that avoided them to be on average, at the end, the same or better accuracy than the vanilla models, but getting a topology smaller than the original. Despite this fact, the models usually reduce their error by getting the same or better values than their counterparts during the next phases.

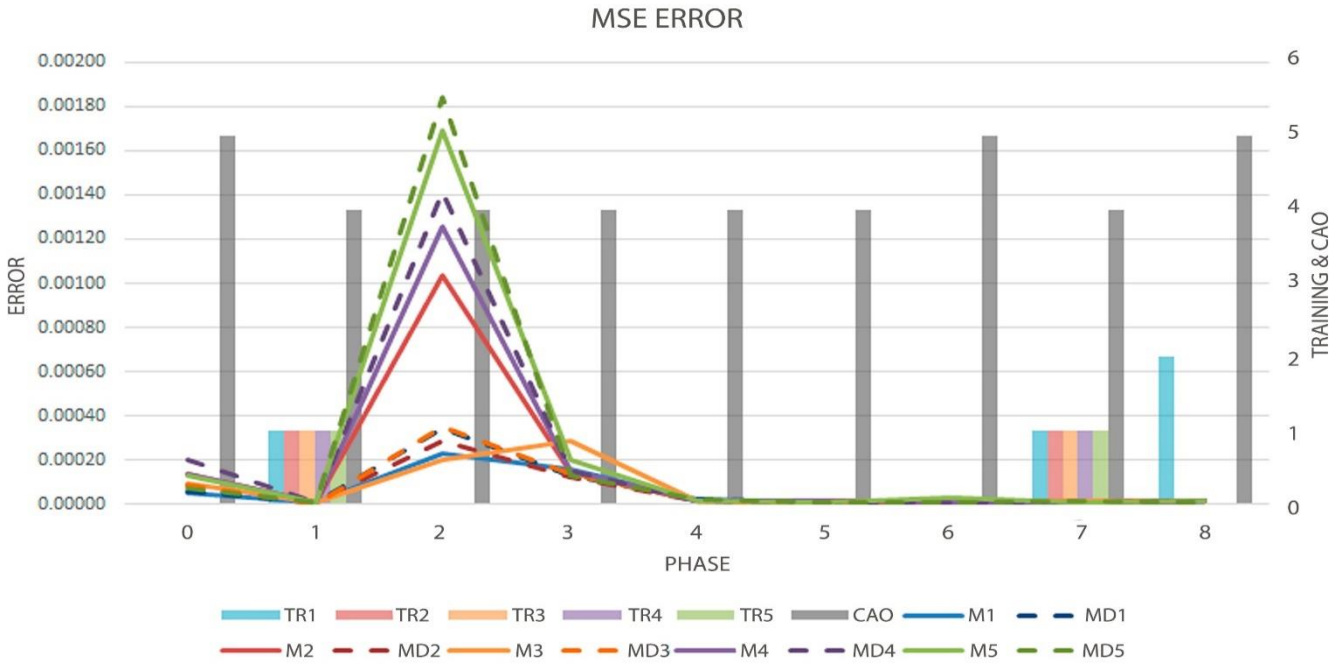

Figure 8. Graph of the MSE metric (Solid and Dotted Lines, measured by left axis), Order evolution (CAO), and the decisions made by the algorithm (TR-Bars, the last two metrics measured by the right axis) in test eleven.

Table 8. Series 11 (MSE), order [Cao], and comparison between the 5 models [M] and their respective dynamic versions [MD], rate, and the decision made [TR] (Growth = 2, Pruning = 1 Normal = 0).

MSE MODEL	Phase 0	Phase 1	Phase 2	Phase 3	Phase 4	Phase 5	Phase 6	Phase 7	Phase 8	AVG
M1	**0.00005**	0.00001	**0.00023**	0.00015	0.00002	0.00001	0.00001	0.00001	0.00001	**0.00006**
MD1	0.00006	0.00001	0.00035	**0.00013**	0.00002	0.00001	0.00001	0.00001	0.00001	0.00007
TR1	0	1	0	0	0	0	0	1	2	
RATE 1	1	0.85	0.85	0.85	0.85	0.85	0.85	0.7	0.85	
M2	0.00014	0.00001	0.00104	0.00014	0.00002	0.00001	0.00002	0.00001	0.00002	0.00015
MD2	**0.00008**	0.00001	**0.00029**	**0.00012**	0.00002	0.00001	**0.00001**	0.00001	**0.00001**	**0.00006**
TR2	0	1	0	0	0	0	0	1	0	
RATE 2	1	0.7	0.7	0.7	0.7	0.7	0.7	0.55	0.55	
M3	0.00009	0.00000	**0.00020**	0.00029	0.00002	0.00001	0.00001	0.00001	0.00001	0.00007
MD3	0.00009	0.00000	0.00035	**0.00014**	0.00002	0.00001	0.00001	0.00001	0.00001	0.00007
TR3	0	1	0	0	0	0	0	1	0	
RATE 3	1	0.55	0.55	0.55	0.55	0.55	0.55	0.4	0.4	
M4	**0.00014**	0.00001	**0.00126**	0.00015	**0.00001**	0.00001	0.00001	0.00001	0.00001	**0.00018**
MD4	0.00020	0.00001	0.00141	0.00015	0.00002	0.00001	0.00001	0.00001	0.00001	0.00020
TR4	0	1	0	0	0	0	0	1	0	
RATE 4	1	0.7	0.7	0.7	0.7	0.7	0.7	0.55	0.55	
M5	0.00013	0.00000	**0.00169**	0.00020	0.00002	0.00001	0.00003	0.00001	0.00001	0.00023
MD5	**0.00008**	0.00001	0.00184	**0.00013**	0.00002	0.00001	**0.00001**	0.00002	0.00001	0.00023
TR5	0	1	0	0	0	0	0	1	0	
RATE 5	1	0.55	0.55	0.55	0.55	0.55	0.55	0.4	0.4	
CAO	5	4	4	4	4	4	5	4	5	

Table 9. Metric (OWA). Final percentage (RATE) of the topology with respect to the initial model. (New units) Number of extra units added, test eleven.

Model	OWA	RATE	NEW Units
M1/MD1	1.11	0.85	0
M2/MD2	**0.54**	0.55	0
M3/MD3	**0.93**	0.4	0
M4/MD4	1.03	0.55	0
M5/MD5	**0.87**	0.4	0

Other particularity presented in the experiments can be seen in tests 5 and 9, where the first phases, didn't present changes of order as can be seen in Table 6, when this happens, the models can present the issue to be the same as their vanilla version, that's why this issue was addressed using the normal phase training, giving the model the capability to manage the old knowledge and be able to distribute the new information over the whole architecture avoiding changing their topology. This implementation gave an improve over 40% in test 5 on models 2, 3, 4, and 5. Even though the dynamic model 1 didn't present an accuracy improvement w.r.t Vanilla, it stays near the same performance as its counterpart. One of these results can be seen in Figures 9 and 10 and Table 10, where it is presented the evolution of the MSE error and the rate of the architecture. Exemplifying the point described above, where architecture modifications are only seen in the last phases.

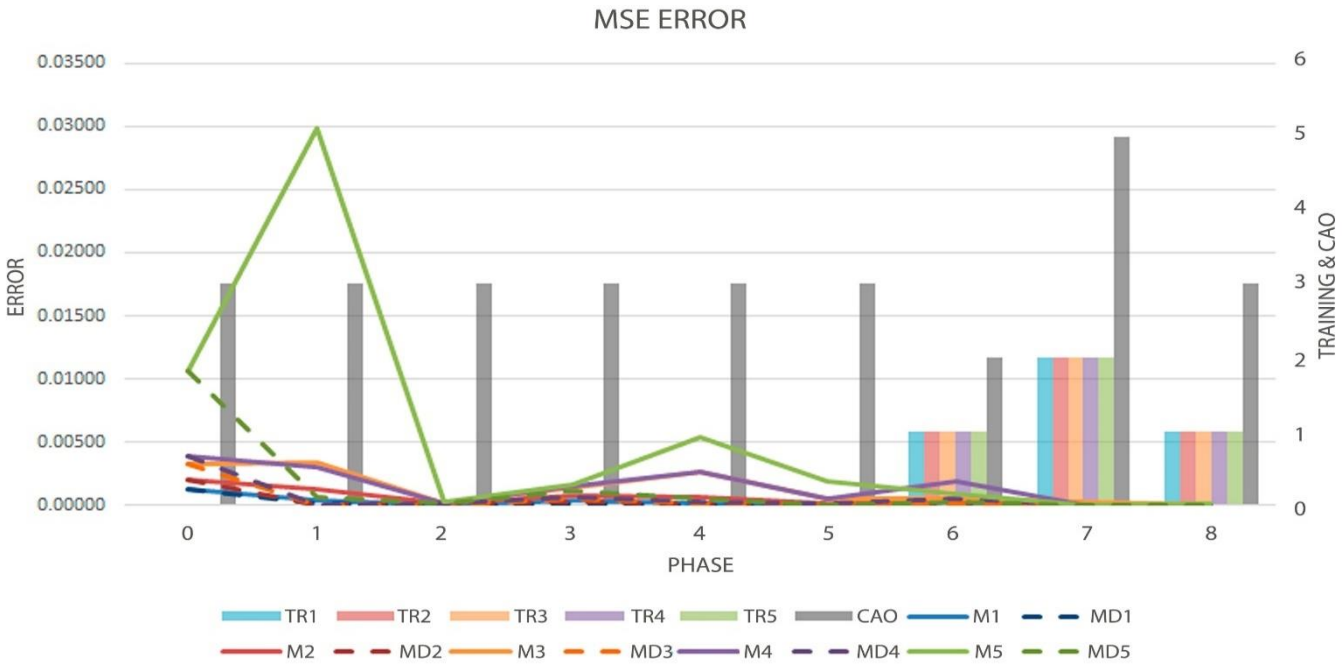

Figure 9. Graph of the MSE metric (Solid and Dotted Lines, measured by left axis), Order evolution (CAO), and the decisions made by the algorithm (TR-Bars, the last two metrics measured by the right axis) in test five.

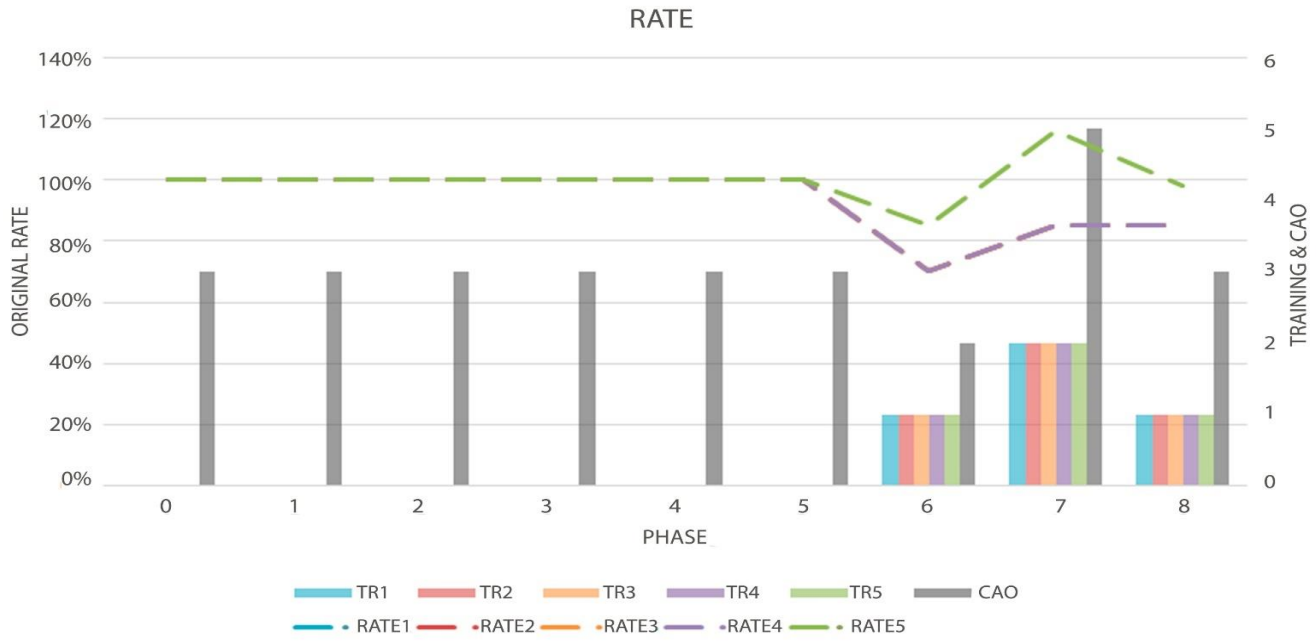

Figure 10. Graph of the Topology Rate (Dotted Lines, measured by left axis), Order evolution (CAO), and the decisions made by the algorithm (TR-Bars, the last two metrics measured by the right axis) in test five.

This process also helped in the situation described before, when a phase after a change of topology doesn't get a good result. Being this training a way to reorganize the knowledge of the model and give better results in the next phases. This case is presented in Table 11, that is one of the results of series 4, which has a reduction of order in phase 1 and after this, the next 3 phases have an increase of order. As can be seen in Table 11 model 3 failed in phases 2 and 3, but even though this occurred the model achieved at the end a 9% improvement, this result can be seen in Table 12.

Table 10. Series 5 (MSE), order [Cao] and comparison between the 5 models [M] and their respective dynamic versions [MD], rate, and the decision made [TR] (Growth = 2, Pruning = 1 Normal = 0).

MSE MODEL	Phase 0	Phase 1	Phase 2	Phase 3	Phase 4	Phase 5	Phase 6	Phase 7	Phase 8	AVG
M1	**0.00123**	0.00034	0.00007	0.00034	0.00017	**0.00003**	0.00027	0.00003	**0.00001**	0.00028
MD1	0.00128	**0.00017**	**0.00006**	**0.00015**	**0.00005**	0.00004	**0.00007**	**0.00002**	0.00005	**0.00021**
TR1	0	0	0	0	0	0	1	2	1	
RATE 1	1	1	1	1	1	1	0.7	0.85	0.85	
M2	0.00196	0.00131	**0.00008**	0.00081	0.00061	0.00012	0.00022	**0.00002**	0.00001	0.00057
MD2	**0.00195**	**0.00005**	0.00009	**0.00029**	**0.00012**	**0.00007**	**0.00008**	0.00009	**0.00001**	**0.00031**
TR2	0	0	0	0	0	0	1	2	1	
RATE 2	1	1	1	1	1	1	0.7	0.85	0.85	
M3	0.00331	0.00339	0.00008	0.00134	0.00259	0.00054	0.00050	0.00026	0.00004	0.00134
MD3	**0.00326**	**0.00005**	**0.00007**	**0.00046**	**0.00015**	**0.00009**	**0.00015**	**0.00003**	**0.00001**	**0.00048**
TR3	0	0	0	0	0	0	1	2	1	
RATE 3	1	1	1	1	1	1	0.7	0.85	0.85	
M4	0.00389	0.00307	0.00010	0.00146	0.00267	0.00057	0.00192	0.00002	0.00002	0.00152
MD4	**0.00386**	**0.00005**	**0.00008**	**0.00064**	**0.00029**	**0.00016**	**0.00055**	0.00002	**0.00001**	**0.00063**
TR4	0	0	0	0	0	0	1	2	1	
RATE 4	1	1	1	1	1	1	0.7	0.85	0.85	
M5	0.01068	0.02985	0.00021	0.00160	0.00532	0.00189	0.00082	0.00002	0.00008	0.00561
MD5	**0.01063**	**0.00061**	**0.00012**	**0.00113**	**0.00056**	**0.00004**	**0.00027**	0.00002	**0.00002**	**0.00149**
TR5	0	0	0	0	0	0	1	2	1	
RATE 5	1	1	1	1	1	1	0.85	1.16	0.98	
CAO	3	3	3	3	3	3	2	5	3	

Table 11. Series 4 (MSE), order [Cao], and comparison between the 5 models [M] and their respective dynamic versions [MD], rate, and the decision made [TR] (Growth = 2, Pruning = 1 Normal = 0).

MSE MODEL	Phase 0	Phase 1	Phase 2	Phase 3	Phase 4	Phase 5	Phase 6	Phase 7	Phase 8	AVG
M1	**0.00047**	0.00216	0.00051	0.00008	0.00008	0.00007	0.00015	0.00022	0.00003	0.00042
MD1	0.00055	**0.00162**	**0.00017**	**0.00007**	**0.00003**	**0.00002**	**0.00002**	**0.00014**	**0.00002**	**0.00029**
TR1	0	1	2	2	2	0	0	0	0	
RATE 1	1	0.85	1.02	1.05	1.07	1.07	1.07	1.07	1.07	
M2	0.00051	0.00328	**0.00035**	0.00016	**0.00015**	0.00018	0.00002	0.00018	0.00003	**0.00054**
MD2	**0.00048**	**0.00287**	0.00059	**0.00006**	0.00075	**0.00005**	0.00002	**0.00015**	**0.00002**	0.00055
TR2	0	1	2	2	2	0	0	0	0	
RATE 2	1	0.7	1.04	1.09	1.19	1.19	1.19	1.19	1.19	
M3	0.00047	0.00281	**0.00012**	0.00006	**0.00005**	0.00005	0.00002	0.00017	0.00002	0.00042
MD3	**0.00046**	**0.00213**	0.00033	0.00006	0.00009	**0.00002**	0.00002	**0.00016**	0.00002	**0.00037**
TR3	0	1	2	2	2	0	0	0	0	
RATE 3	1	0.7	1.06	1.19	1.32	1.32	1.32	1.32	1.32	
M4	0.00059	0.00447	**0.00011**	0.00020	0.00017	0.00004	0.00004	0.00026	0.00003	0.00066
MD4	**0.00057**	**0.00403**	0.00077	**0.00008**	**0.00003**	**0.00003**	**0.00002**	**0.00014**	**0.00002**	**0.00063**
TR4	0	1	2	2	2	0	0	0	0	
RATE 4	1	0.85	1.13	1.23	1.43	1.43	1.43	1.43	1.43	
M5	0.00194	**0.00326**	**0.00057**	**0.00011**	0.00030	0.00030	0.00006	0.00026	0.00003	0.00076
MD5	**0.00180**	0.00349	0.00081	0.00012	**0.00017**	**0.00006**	**0.00003**	**0.00016**	**0.00003**	**0.00074**
TR5	0	1	2	2	2	0	0	0	0	
RATE 5	1	0.85	1.16	1.32	1.65	1.65	1.65	1.65	1.65	
CAO	4	1	3	4	5	5	5	4	5	

Unifying the three methods of training (pruning, growing, and normal) of the proposed system, makes the models able to identify a global trend of the data and get better results in later phases. Figure 11 shows the evolution of models 1 (Vanilla/DyLSTM) over phases 1 to 8 of series 4. This figure presents how model 1 trained with the proposed method acquires better results and anticipates trend changes meanwhile more phases have been processed.

Table 12. Metric (OWA). Final percentage (RATE) of the topology with respect to the initial model. (New units) Number of extra units added, test four.

Model	OWA	RATE	New Units
M1/MD1	**0.65**	1.07	3
M2/MD2	**0.99**	1.19	4
M3/MD3	**0.91**	1.32	5
M4/MD4	**0.93**	1.43	5
M5/MD5	**0.89**	1.65	4

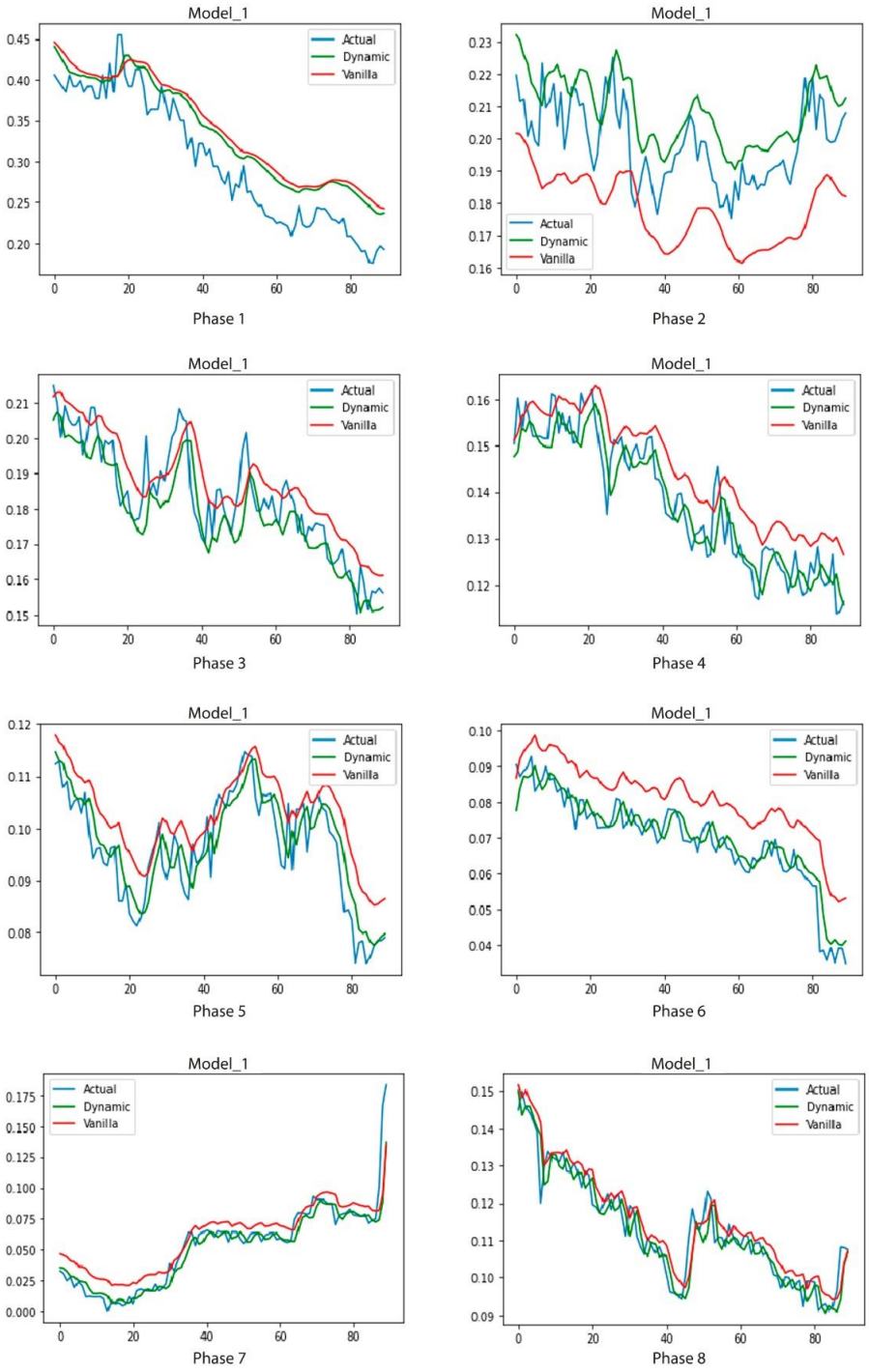

Figure 11. Results of model 1 (Vanilla/DyLSTM) phases 1 to 8 in test four. Blue is the Actual timesereis, green is our proposed method and red is the vanilla model.

6. Conclusions and Future Work

In this work it was presented an algorithm that provides plasticity to recurrent neural networks, using as an example LSTM models applied to the time series forecasting, implementing (1) a growing paradigm based on gradients, (2) a pruning based on magnitudes, and (3) a reorganized knowledge normal phase, implemented through the use of decision policies, based on the calculation of the order of the data to train in each retraining process, when the database is updated.

Making use of some of the data series of the M4 competition, with 10 series of 1000 samples and 4 series with 4000 samples, where almost all the cases and models presented performance improvements by reducing the error function with respect to a static structure. Being the best result an improvement of around 40% between the 5 models and individually the best performance was obtained by the fifth model in series 5 with a 62% of improvement. Obtaining on average an improvement of up to 18%. In the cases that did not present considerable improvements, a similarity in the range of the error to its static counterpart was maintained and despite this, in the cases, where a reduction of order was presented, their architecture was notably reduced. As future work, the presented methodology is going to be improved by exploring new decision-making policies that allows their automatic updating based on the results obtained at each phase. It also looks forward implement a parallel process for predicting the order of the following data sequence, making the system more robust by giving it more information and anticipating the future behavior of the data. Furthermore, we will analyze the possibility to extend our approach by implementing the proposed method in other neural network models.

Author Contributions: Conceptualization, N.A.-D.; methodology, N.A.-D., D.S.N. and C.V.; software, D.S.N.; validation, C.V. and C.L.-F.; formal analysis, N.A.-D.; investigation, N.A.-D., D.S.N. and C.V.; resources, Carlos López-Franco.; writing—D.S.N.; writing—review and editing, N.A.-D. and C.V.; visualization, D.S.N.; supervision, N.A.-D. and C.V.; project administration, N.A.-D. and C.L.-F.; funding acquisition, C.L.-F. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by CONACYT México grant number CB258068.

Data Availability Statement: Publicly available dataset was analyzed in this study. This data can be found here: [25].

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Slawek, S. A hybrid method of exponential smoothing and recurrent. *Int. J. Forecast.* **2020**, *36*, 75–85.
2. Khaldi, R.; El Afia, A.; Chiheb, R.; Tabik, S. What is the best RNN-cell structure to forecast each time series behavior? *Expert Syst. Appl.* **2023**, *215*, 119–140. [CrossRef]
3. Peixeiro, M. *Time Series Forecasting in Python*, 1st ed.; Manning: New York, NY, USA, 2022; pp. 287–299.
4. Haykin, S. *Neural Networks and Learning Machines*, 3rd ed.; Pearson: Hamilton, ON, Canada, 2009; pp. 1–45.
5. Vapnik, V. Principles of Risk Minimization for Learning Theory. *Adv. Neural Inf. Process.* **1992**, *4*, 831–838.
6. Byoung-Tak, Z. An Incremental Learning Algorithm That Optimizes Network Size and Sample Size in One Trial. *IEEE Int. Conf. Neural Netw.* **1991**, *1*, 215–220.
7. Baum, E.; Haussler, D. What Size Net Gives Valid Generalization? *Neural Comput.* **1989**, *1*, 151–160. [CrossRef]
8. Rehmer, A.; Kroll, A. On the vanishing and exploding gradient problem in Gated Recurrent Units. *IFAC-PapersOnLine* **2020**, *53*, 1243–1248. [CrossRef]
9. Tsay, R.S.; Chen, R. *Nonlinear Time Series Analysis*, 1st ed.; Wiley: Hoboken, NJ, USA, 2019; pp. 173–184.
10. Dai, X.; Yin, H.; Jha, N.K. Grow and Prune Compact, Fast and Accurate LSTMs. *IEEE Trans. Comput.* **2020**, *69*, 441–452. [CrossRef]
11. Blalock, D.; Gonzalez Ortiz, J.J.; Frankle, J.; Guttag, J. What is the State of Neural Network Pruning? In Proceedings of the 3rd MLSys Conference, Austin, TX, USA, 2–4 March 2020.
12. Fritzke, B. Growing cell structures—A self-organizing network for unsupervised and supervised learning. *Neural Netw.* **1994**, *7*, 1441–1460. [CrossRef]
13. Benitez, E.M.S.; Pérez, M.Q.L. Plasticidad cerebral, una realidad neuronal. *Rev. De Cienc. Médicas De Pinar Del Río* **2019**, *23*, 599–609.

14. Cao, L. Practical method for determining the minimum embedding dimension of a scalar time series. *Elsevier Sci.* **1997**, *110*, 43–50. [CrossRef]
15. Kennel, M.B.; Brown, R.; Abarbanel, H.D. Determining embedding dimension for phase-space reconstruction using a geometrical construction. *Am. Phys. Soc.* **1992**, *45*, 3403–3411. [CrossRef] [PubMed]
16. Makridakis, S.; Spiliotis, E.; Assimakopoulos, V. The M4 Competition: 100,000 time series and 61 forecasting methods. *Int. J. Forecast.* **2020**, *36*, 54–74. [CrossRef]
17. Sanchez, E.N.; Felix, R.A. Nonlinear identification via variable structure recurrent neural networks. In Proceedings of the 15th Triennial World Congress, Barcelona, Spain, 21–26 July 2002.
18. Tabares, H.; Branch, J.; Valencia, J. Generación dinámica de la topología de una red neuronal artificial del tipo perceptron multicapa, Revista Facultad de Ingenieria. *Univ. De Antioq.* **2006**, *38*, 146–162.
19. Alam, K.M.R.; Siddique, N.; Adeli, H. A dynamic ensemble learning algorithm for neural networks. *Neural Comput. Appl.* **2019**, *32*, 8675–8690. [CrossRef]
20. Dai, X.; Yin, H.; Jha, N.K. Incremental Learning Using a Grow and Prune Paradigm with Efficient Neural Networks. *IEEE Trans. Emerg. Top. Comput.* **2020**, *10*, 752–762. [CrossRef]
21. Chollet, F. *Deep Learning with Python*; Manning Publications Co.: Shelter Island, NY, USA, 2018; pp. 202–216.
22. Olah, C. Github-Understanding LSTM Networks. 27 August 2015. Available online: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (accessed on 20 October 2022).
23. Denil, M.; Shakibi, B.; Dinh, L.; Ranzato, M.A.; De Freitas, N. Predicting Parameters in Deep Learning. *Adv. Neural Inf. Process. Syst.* **2013**, *2*, 2148–2156.
24. Spiliotis, E.; Kouloumos, A.; Assimakopoulos, V.; Makridakis, S. Are forecasting competitions data representative of the reality? *Int. J. Forecast.* **2020**, *1*, 37–53. [CrossRef]
25. Yogesh, S. Kaggle-M4 Forecasting Competition Dataset. 2020. Available online: https://www.kaggle.com/datasets/yogesh94/m4-forecasting-competition-dataset (accessed on 21 October 2022).