Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche

researchportal.unamur.be

University of Namur

THESIS / THÈSE

MASTER IN COMPUTER SCIENCE PROFESSIONAL FOCUS IN DATA SCIENCE

Neuroevolutionary Transfer Learning method for Time Series Predictions

VELLINGER, Aymeric

Award date:
2023

Awarding institution:
University of Namur

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 08. Sep. 2023

UNIVERSITÉ DE NAMUR
Faculté d'informatique
Année académique 2022-2023

Neuroevolutionary Transfer Learning method for Time Series Predictions

Aymeric Vellinger

.... (Signature pour approbation du dépôt - REE art. 40)

Promoteur : Wim Vanhoof

Mémoire présenté en vue de l'obtention du grade de Master 120 en Sciences Informatiques

à finalité spécialisée en Data Science

Faculté d'Informatique – Université de Namur

RUE GRANDGAGNAGE, 21 ● B-5000 NAMUR(BELGIUM)

Acknowledgements

I would like to express my heartfelt gratitude in writing this Master's thesis. I am deeply thankful to the University of Namur for providing me with the opportunity to pursue my studies and conduct research in such a supportive academic environment.

I am particularly grateful to Professor Wim Vanhoof for his invaluable guidance, expertise, and unwavering support throughout the entire process. His insightful feedback and encouragement have been instrumental in shaping the outcome of this thesis.

I would also like to extend my sincere appreciation to the University Pablo de Olavide, where I had the privilege to carry out my internship. I deeply appreciate the incredible support and unparalleled kindness of Professor Divina during his supervision at the University Pablo de Olavide. His guidance, expertise, and unwavering encouragement have been instrumental in shaping the outcome of this thesis. I am immensely grateful for the opportunity to work under his supervision and for the valuable insights and experiences gained through his mentorship.

Furthermore, I would like to express my gratitude to my colleagues in the laboratory who have been instrumental in assisting me with this project. Their collaboration, support, and valuable discussions have played a vital role in the success of this research endeavor.

Finally, I am indebted to all the individuals who have directly or indirectly contributed to this thesis. Your assistance, encouragement, and inspiration have been invaluable, and I am truly grateful for the opportunity to learn and grow as a researcher during this journey.

Thank you all for your unwavering support, guidance, and belief in my abilities.

Résumé

Nous avons développé une technique de neuroévolution spécifique pour améliorer les réseaux LSTM. Notre approche utilise une méthode basée sur la grammaire pour faire évoluer les réseaux LSTM dans le contexte de la prédiction de séries temporelles, en s'appuyant sur une technique précédemment utilisée pour les réseaux CNN.

Pour accélérer les calculs, nous avons intégré l'apprentissage par transfert à notre méthode. Nous avons évalué les performances de notre approche en la comparant à d'autres techniques de pointe en matière de prévision de séries temporelles. Nous avons utilisé vingt séries temporelles contenant des données collectées à partir de capteurs placés sur des porcs ibériques.

Les résultats obtenus confirment l'efficacité de notre stratégie dans ce domaine. Globalement, notre proposition démontre le potentiel de produire des modèles d'apprentissage profond précis et efficaces pour la prédiction de séries temporelles, ainsi que l'adaptabilité de l'apprentissage par transfert à de nouveaux ensembles de données.

Mots-clés : *Prévision de séries temporelles, Neuroévolution, Apprentissage profond*

1 Abstract

We propose a neuroevolution technique specifically designed for evolving LSTM networks. The proposed technique uses a grammar-based approach to evolve LSTM neural networks for time series prediction tasks, and is based on a previous techniques which was designed in order to evolve CNN networks.

We use transfer learning in order to reduce the computational time of our proposal. We have compared results obtained by our proposal with other state of the art time series forecasting techniques on twenty time series, which contains data generated by sensors placed on a number of Iberian pigs. Results obtained confirm the effectiveness of the strategy proposed in this work.

Overall, we showcase the potential of our proposal in producing precise and efficient deep learning models for time series prediction, as well as the adaptability of transfer learning to new datasets.

Keywords : *Time series forecasting , Neuroevolution , Deep Learning*

Table des matières

1 Abstract **2**

2 State of the art **5**
- 2.1 Introduction . . . 5
- 2.2 Background . . . 5
  - 2.2.1 LSTM . . . 5
  - 2.2.2 Genetic Algorithm (GA) . . . 6
  - 2.2.3 Evolutionary Strategy (ES) . . . 8
  - 2.2.4 Vanilla Grammatical evolution (GE) . . . 9
  - 2.2.5 Structured Grammatical evolution (SGE) . . . 9
  - 2.2.6 Dynamic Structured Grammatical Evolution (DSGE) . . . 10
- 2.3 Hybrid Technique for hyperparameter optimization . . . 11
  - 2.3.1 Deep evolutionary network structured representation . . . 11
  - 2.3.2 Comparison with random search and grid search . . . 11
  - 2.3.3 Comparison with Bayesian optimization [25] . . . 13

3 Research **15**
- 3.1 Problematic . . . 15
- 3.2 Tools and Methodology . . . 15
  - 3.2.1 Environement . . . 15
  - 3.2.2 Datasets . . . 16
- 3.3 Grammar . . . 19
- 3.4 Neuroevolution . . . 25
  - 3.4.1 Crossover . . . 25
  - 3.4.2 Mutation . . . 26
  - 3.4.3 Evaluation . . . 26
- 3.5 Transfer Learning . . . 26
- 3.6 Application of the methodology . . . 27
- 3.7 Comparison . . . 32
  - 3.7.1 Gaussian process : . . . 32
  - 3.7.2 Gradient boosting : . . . 35
  - 3.7.3 Random Forest : . . . 38
  - 3.7.4 Light GBM : . . . 41
  - 3.7.5 Non Optimized LSTM . . . 41
  - 3.7.6 Result of the comparison . . . 44

4 Discussion **46**

Bibliography **50**

Acronymes

GA Genetic algorithm. 6, 11, 21, 24, 25

ES Evolutionary Strategy. 8, 9, 24, 27

LSTM Long Short Term Memory. 1-45

CNN Convolutional Neural Network. 1, 12

ANN Artificial Neural Network. 21

RNN Recursive Neural Network. 5, 27

GE Grammatical Evolution. 9, 11

SGE Structured Grammatical Evolution. 9, 10, 11, 15

DSGE Dynamic Structured Grammatical Evolution. 10, 11, 18, 21, 24, 25

GP Gaussian Process. 6, 31

GB Gradient Boosting. 34

LightGB(M) Light Gradient Boosting Machine. 28, 40

2 State of the art

2.1 Introduction

Hyperparameter optimization is a crucial step in the training of neural networks as it determines the overall performance of the network. Hyperparameters control various aspects of the network, such as the learning rate, number of hidden layers, and regularization strength, among others. Selecting appropriate hyperparameters can significantly impact the accuracy and efficiency of the network. However, hyperparameter optimization is a challenging task due to the vast number of hyperparameters involved and the high dimensionality of the search space. Traditional methods, such as grid search and random search, can be computationally expensive and may not result in the best hyperparameter configuration. To address these challenges, researchers have proposed various methods for hyperparameter optimization in neural networks, including Bayesian optimization, grid-search, and evolutionary algorithms, among others. These methods aim to find the optimal set of hyperparameters that result in the best performance of the network. In the last few years, evolutionary algorithms have had a lot of progress and success, especially in incorporating hybrid methods for hyperparameters optimization in convolutional neural network (CNN).

In this state of the art, we will focus on the use of evolutionary algorithms for hyperparameter optimization in LSTMs, and demonstrate that it can lead to significant improvements in model performance and provide a more efficient and effective way to select optimal hyperparameters.

2.2 Background

2.2.1 LSTM

Deep neural networks (DNNs) have been widely adopted in recent years with major advances allowing their integration into professional use. Nowadays, recursive neural networks (RNN) overperform DNNs in a wide variety of problems and have found particular attention in the research world. However, RNNs face the problem of gradient fading when learning to remember past events. To overcome this difficulty, the Long Short-Term Memory network, often known as LSTM [1], has attracted a lot of interest recently. LSTM is an RNN architecture created to more precisely characterize temporal sequences and their long-range dependencies.

It achieved outstanding outcomes for a number of time-series challenges, including named entity recognition [2], chunking [3], acoustic modeling [4], and Part-of-Speech (PoS) tagging [5], to mention a few. The method is nevertheless plagued by a well-known issue with neural network-based methods : the correct selection of its hyperparameters, often left to human appreciation. The main method used for optimizing algorithmic parameterisations is grid search, which performs an exhaustive search of the discrete parameter's domain. Yet, a lot of computing power is needed because the grid's parameter combinations must all be verified.

The search for strategies to automatically configure ML algorithms, is not a new concept, and is referred as Automated machine learning (AutoML). The main objective of these methods is to build and configure ML systems without the assistance of humans. By doing so, the burden of an exhaustive iterative trial-and-error search would be reduced and the requirement for subject knowledge and experience would be minimized.

Nowaday, many works addressed the problem of hyperparameter fine-tuning in AutoML using nature-inspired metaheuristic optimization techniques. For evolving a neural network using a genetic algorithm, two main approaches are used. In the first approach, the network topology and the network weights are evolved simultaneously [6], while in the second approach

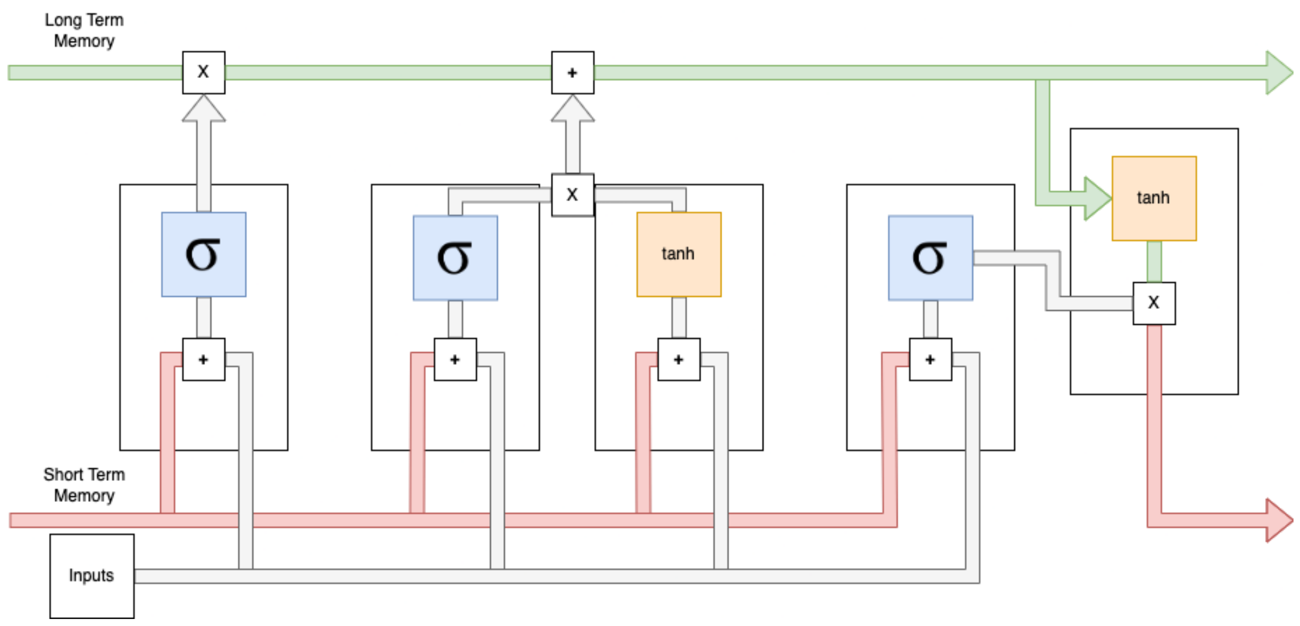


FIGURE 2.1 – Long Short Term Memory Architecture

only the network topology is evolved and the parameters are estimated using a standard approach such as a gradientbased optimization method [7]. The representation of the neural network is another important aspect of the evolutionary approach. In the first, a binary coding of the topology and weights is used. In the second approach, a table structure is used to represent the connectivity between the neurons and the network's weights.

The work conducted by Valeria V. Krzhizhanovskaya et al. [8] compares the performance of four well-known evolutionary optimization algorithms, i.e., GA, GP, GSGP, and SGP, as well as a random search, to the task of LSTM network hyperparameter fine-tuning in the context of PoS tagging.

From these results, it is possible to extract two main conclusions : metaheuristic optimization techniques performed better than a random search concerning the three datasets, which validates the employment of such methods for the task ; the tree-based algorithms, i.e., GP, GSGP, and SGP, performed better than GA in two out of three datasets. Such behavior is expected since these techniques employ more robust approaches to solve the optimization problem. Experiments conducted over three well-known public datasets confirmed the effectiveness of the proposed approach since all four metaheuristic algorithms outperformed the random search. Further, although GA presented a smoother convergence during the optimization steps, the tree-based evolutionary techniques obtained the best loss and accuracy results when considering a more extended training period over the testing sets.

2.2.2 Genetic Algorithm (GA)

Genetic Algorithm (GA) is one of the first population-based stochastic algorithm proposed in the history [9]. Similar to other EAs, the main operators of GA are selection, crossover, and mutation :

— Selection rules select the individuals, called parents, that contribute to the population at the next generation. The selection is generally stochastic, and can depend on the individuals' scores.

— Crossover rules combine two parents to form children for the next generation.

— Mutation rules apply random changes to individual parents to form children.

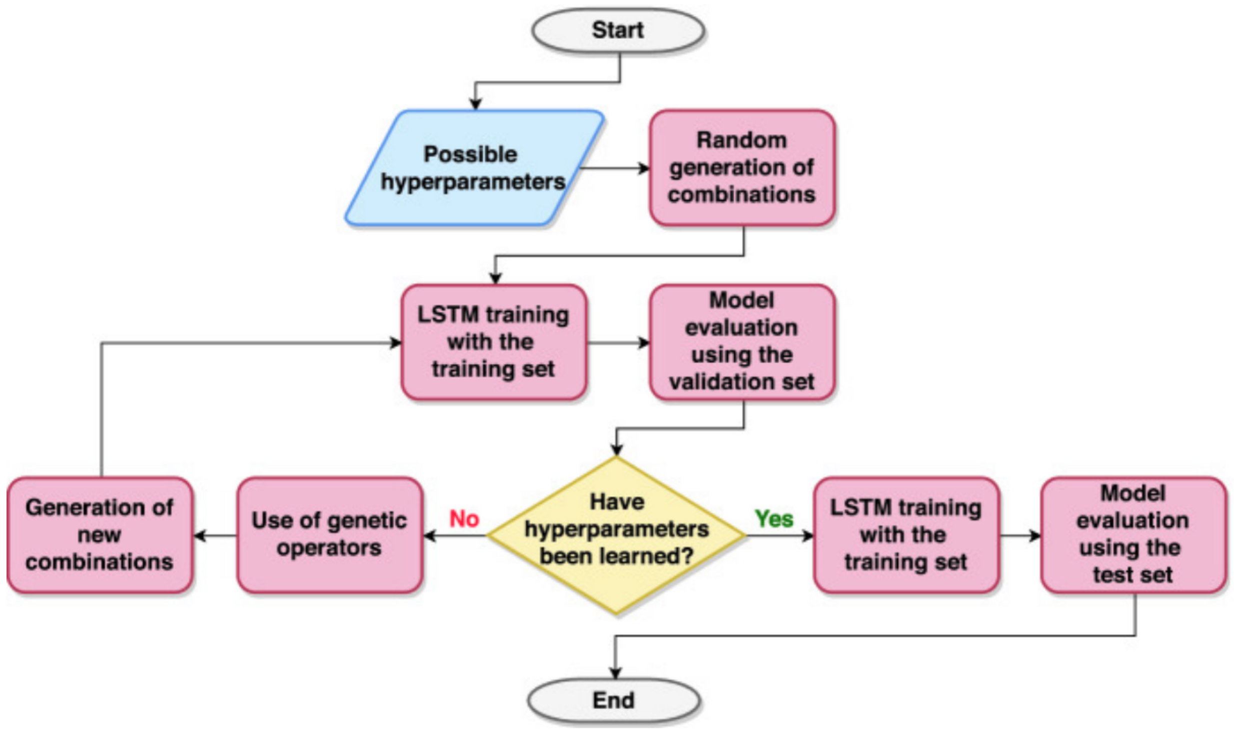

Start
Possible hyperparameters
Random generation of combinations
LSTM training with the training set
Model evaluation using the validation set
Have hyperparameters been learned?
No
Yes
Use of genetic operators
Generation of new combinations
LSTM training with the training set
Model evaluation using the test set
End

FIGURE 2.2 – Pipeline for fine-tuning hyperparameters, from Valeria V. Krzhizhanovskaya et al. [8]

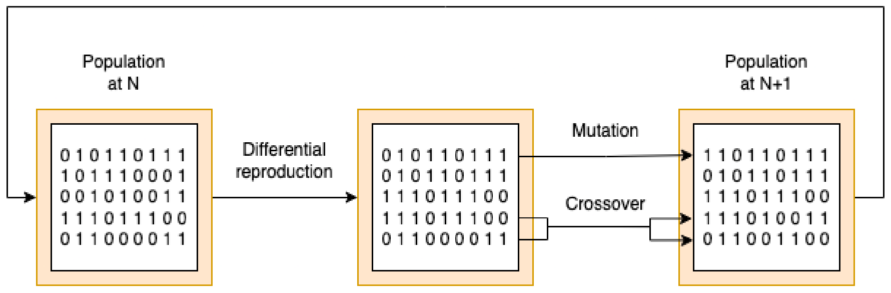


FIGURE 2.3 – Genetic Algorithm overview

Work on the application of genetic algorithm with different metaheuristics was conducted by Nikolaos Gorgolis et al. [10] They ran tests on the Tensorflow [11] multilayered LSTM consisting of an input layer followed by an embedding layer, two LSTM hidden layers and a dropout layer, finally followed by a dense layer and a Softmax activation layer. A second implementation was made following the keras [12] documentation. Their experiments show that the genetic algorithm consists of an efficient and viable method for fine-tuning a neural network.

Another experiment of the same kind was conducted by Hussain Alibrahim and Simone A. Ludwig [13] The results of the experiments both stated that the genetic algorithm performed better than the grid search and the Bayesian algorithm, and that its resulting configurations outperform the default setting usually proposed [14], [15], while the whole approach offers both affordability and sufficient convergence.

The third approach is considered the most sophisticated since it involves using a higher-level language such as grammar to describe the network topology (Fig.2.5). The latter approach

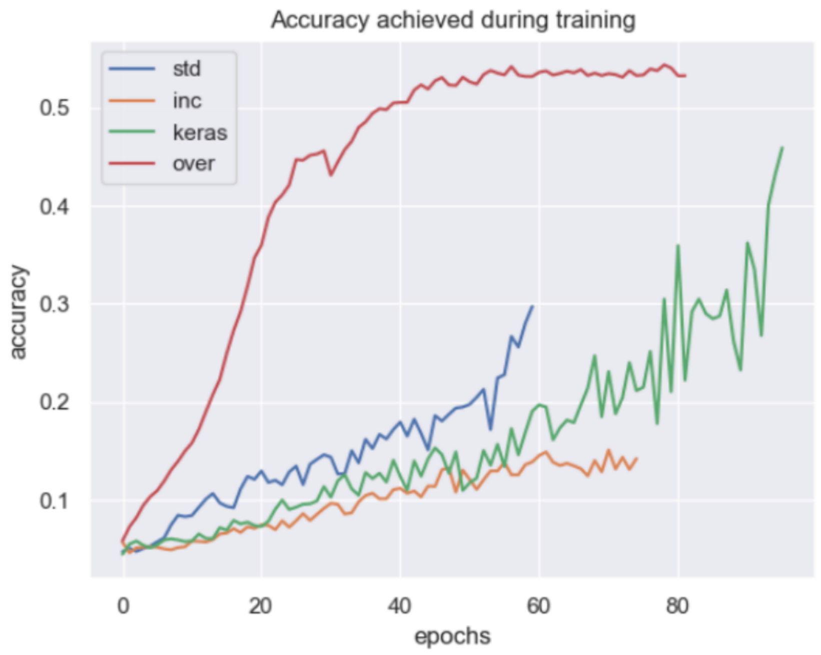


FIGURE 2.4 – Both variants implemented give better results than the default configuration

gives better control to the network architect since he can create node layers and better describe complex topologies.

This method is better known as grammatical evolution (GE) [16] , and has different variants to accomplish the same goal. All methods are based on rewriting systems that are formally defined by Backus Naur Form (BNF) notation for expressing the grammar in the form of production rules i.e by a 4-tuple : G = (N, T, P, S), where :

N is the set of non-terminal symbols ;

T is the set of terminal symbols ;

P is the set of production rules of the form x : :=y, $x \epsilon N$ and $y \epsilon N \cup T*$;

S is the initial symbol.

2.2.3 Evolutionary Strategy (ES)

An evolutionary strategy (ES) is a search algorithm inspired by natural selection that is used to optimize a particular function or set of functions. ES operates by creating a population of candidate solutions or individuals, which are then evaluated based on their fitness or performance with respect to the function being optimized.

During the evaluation process, individuals with higher fitness or better performance are selected to reproduce and create offspring that inherit the characteristics of their parents. The process of reproduction involves genetic operations such as mutation, crossover, and selection, which are based on principles of genetics and evolution.

The offspring then undergo the same evaluation process as their parents, and the cycle continues until a termination criterion is met, such as a maximum number of generations or a satisfactory level of performance.

ES is a type of evolutionary algorithm that is particularly suited to optimizing continuous, high-dimensional, and noisy functions. It has been applied to a wide range of optimization problems in various fields, including engineering, finance, and machine learning.

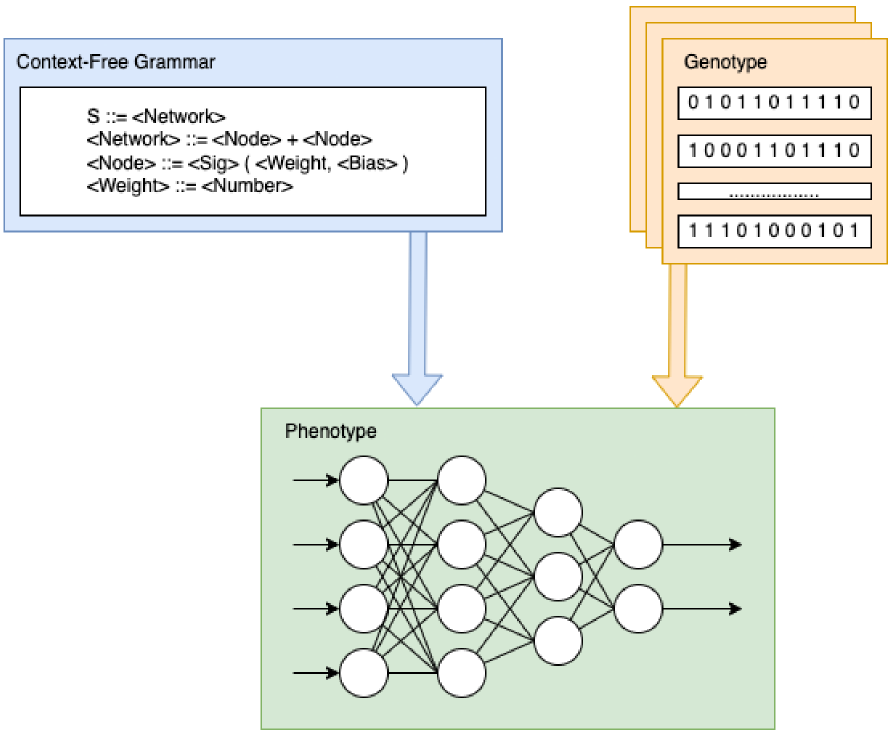


FIGURE 2.5 – Grammatical Evolution

Overall, ES is a powerful and flexible optimization method that has proven to be effective in solving complex problems in many domains.

A deep understanding of the subject and multiple example of usage can be found in the book Evolutionary algorithms in theory and practice : evolution strategies, evolutionary programming, genetic algorithms by Back and Thomas [17]

2.2.4 Vanilla Grammatical evolution (GE)

GE is an evolutionary algorithm that can evolve complete programs in an arbitrary language using a variable-length binary string. e.g, rules for arithmetic expressions could be written as :

$S =< expr >$

$< expr >::=< expr >< op >< expr >$ (0)

$|a$ (1)

$|b$ (2)

$< op >::= +|-|*|/$

These rules can also be represented as a tree. Note that the generale decoding procedure reads each codon from left to right and selects the derivation step to be applied as :

$$rule = codon MOD NonTerminalPossibilities$$

The amount of codons in the genotype may not be sufficient if the language being utilized is recursive. Wrapping is employed to address this problem, by reusing the genetic material and reading the codons from the beginning of the genotypic material.

2.2.5 Structured Grammatical evolution (SGE)

Structured Grammatical Evolution (SGE) [18] is an enhanced genotypic representation for GE. In SGE each gene is linked to a specific non-terminal and is composed by a list of

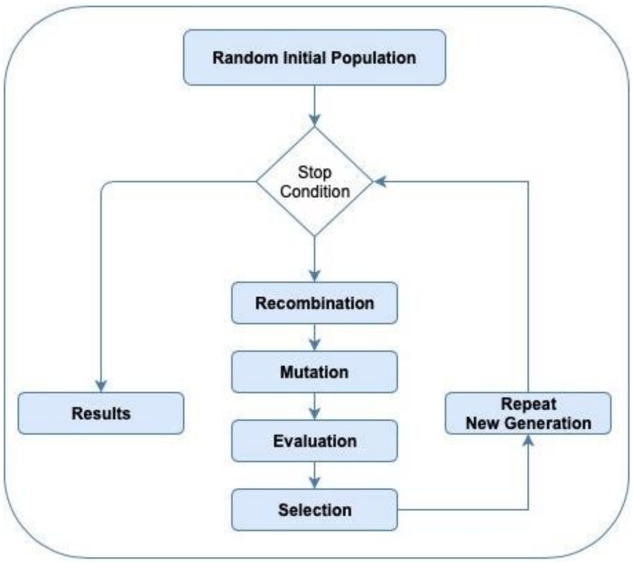


FIGURE 2.6 – Evolution Strategy Flowchart

integers. The length of each list is determined by computing the maximum possible number of expansions of the corresponding non-terminal. The values that are inside the lists correspond to the number of possible expansion choices. Therefore, when performing the mapping it is possible to remove the modulo rule, thus reducing the redundancy associated with it.

By taking the same example :

$S =< expr >$

$< expr >::=< expr >< op >< expr > |a|b$

$< op >::= +|-| * |/$

Looking into the grammar, we see that the $< expr >$ production is recursive. To deal with recursion, a maximum depth is defined, and a set of intermediate symbols are created. Assuming that 2 levels of recursion were defined it becomes :

$S =< expr >$

$< expr >::=< exprlvl0 >< op >< exprlvl0 > | < var >$

$< exprlvl0 >::=< exprlvl1 >< op >< exprlvl1 > | < var >$

$< exprlvl1 >::=< var >< op >< var > | < var >$

$< op >::= +|-| * |/$

$< var >::= a|b$

The decoding procedure follows the same rationale than the one described for GE, with the difference that the codons are not read sequentially from a single list of integers, but are rather read sequentially from the list associated with the nonterminal symbol that is to be expanded.

2.2.6 Dynamic Structured Grammatical Evolution (DSGE)

In SGE the input grammar must be pre-processed so that recursion is removed, and the maximum number of expansion possibilities for each symbol determined. This has been pointed out as a drawback of SGE and to tackle it ,Dynamic Structured Grammatical Evolution (DSGE) was introduced [19]. In DSGE there is no need to pre-process the grammar, as it is expanded on the fly during the evolutionary process. Another advantage of DSGE is that all of the genotype is used. While in GE and SGE the genotype encodes the largest allowed sequence, in DSGE the genotype grows as needed.

The representation is similar to the one used in SGE, with one main difference : instead of computing and generating the maximum number of derivations for each of the grammar's non-terminal symbols, a variable length representation is used, where just the number of needed derivations are encoded. Consequently, there is no need to create intermediate symbols to deal with recursive rules.

2.3 Hybrid Technique for hyperparameter optimization

The optimization of hyperparameters in Long Short Term Memory (LSTM) neural networks is a challenging task, as it requires a careful balancing of various parameters to achieve optimal performance. A proposed approach is the use of Dynamic Structured Grammatical Evolution (DSGE) and Genetic Algorithm (GA) to optimize the hyperparameters of neural networks, combining the representation power of Grammatical Evolution and the optimization capabilities of Genetic Algorithm. The state-of-the-art studies have shown that the hybrid method outperforms traditional grid search and random search approaches in terms of accuracy and efficiency.[20] Overall, the combination of DSGE and GA for hyperparameter optimization in neural networks has been shown to be a promising approach, providing improved accuracy and efficiency compared to traditional methods.

2.3.1 Deep evolutionary network structured representation

Filipe Assunção et al. proposed the DENSER (deep evolutionary network structured representation) method for deep neural network architecture optimization [21]. DENSER uses a combination of Grammatical Evolution and Genetic Algorithm to optimize both the architecture and hyperparameters of deep neural networks. The method leverages the representation power of Grammatical Evolution to model complex network architectures and the optimization capabilities of the Genetic Algorithm to search for optimal solutions. The results of the method show improved performance compared to traditional methods, such as grid search and random search, in terms of accuracy and efficiency. DENSER has been applied to various applications, such as image classification and object detection, and has shown promising results in terms of accuracy and computational efficiency. The method has been evaluated on several benchmark datasets, demonstrating its potential as a valuable tool for deep neural network architecture optimization.

DENSER is a state-of-the-art method for deep neural network architecture optimization that combines the benefits of Grammatical Evolution and Genetic Algorithm. The method has shown improved performance compared to traditional methods, making it a valuable tool for various applications in the field of deep learning.

Filipe Assunção et al. also published Fast-DENSER [22] and Fast-DENSER++ [23] The main difference between Fast-DENSER (and its variant F-DENSER++) and DENSER lies in their computational efficiency. Both architectures are designed to be a more efficient version of DENSER, which was computationally expensive and took a long time to complete the optimization process.

2.3.2 Comparison with random search and grid search

Two widely used methods for hyperparameter optimization are random search and grid search. Random search involves selecting hyperparameters randomly from a predefined distribution, while grid search involves exhaustively testing a fixed set of hyperparameter combinations.

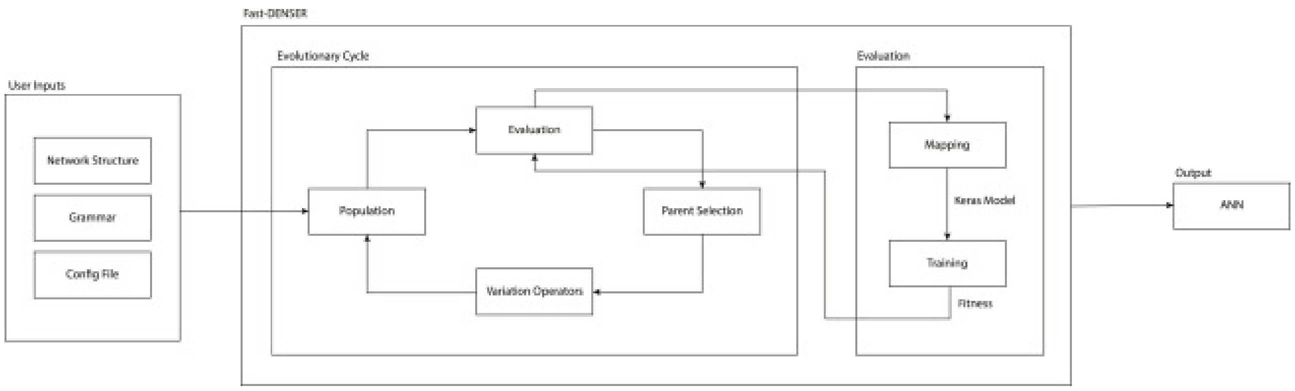


FIGURE 2.7 – the Fast-DENSER architecture proposed by Filipe Assunção et al.

"Random Search for Hyper-Parameter Optimization" by James Bergstra et al. [24] is a paper that explores the effectiveness of random search as a method for optimizing hyperparameters in machine learning algorithms. The paper compares random search with other popular hyper-parameter optimization methods, including grid search and Bayesian optimization, and finds that random search is a competitive and often more efficient method. Thei method has been tested on a sample of the MNIST dataset.

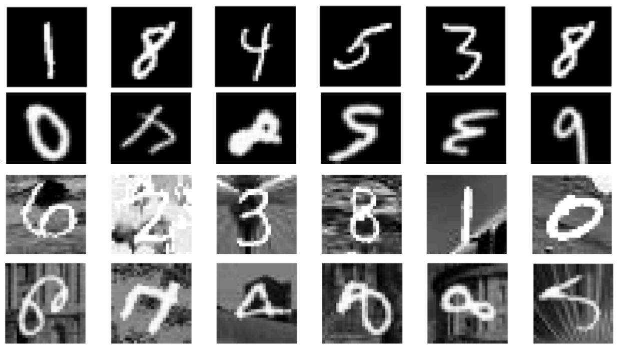


FIGURE 2.8 – From top to bottom, samples from the mnist, mnist rotated, mnist background images, mnist rotated background images data sets.

The MNIST (Modified National Institute of Standards and Technology) dataset is a widely used dataset in the field of machine learning and computer vision. It consists of a large collection of handwritten digits (0 to 9), each represented as a 28x28 grayscale image. Their is also variation in the MNIST dataset :

— MNIST Rotated —the digits are rotated between 0 and 360$\circ$;
— MNIST Background —instead of a clean white background, a realworld image is used as background for the digit ;
— MNIST Rotated + Background —combines the previous two modifcations, i.e., the digits are rotated, and an image is used in the background.

The results in Figure 2.9 shows that Random search is better than grid search with few trials, going to his local maximum faster, but has some difficulties to make some improvement afterward.

As random search is a better option in low dimensional CNN for tasks that don't need complex training, DENSER is more suited for more complex architecture, as it has an accuracies of 97.71%, 98.62%, and 92.66%, for the MNIST Rotated, MNIST Background, and MNIST Rotated + Background, respectively.

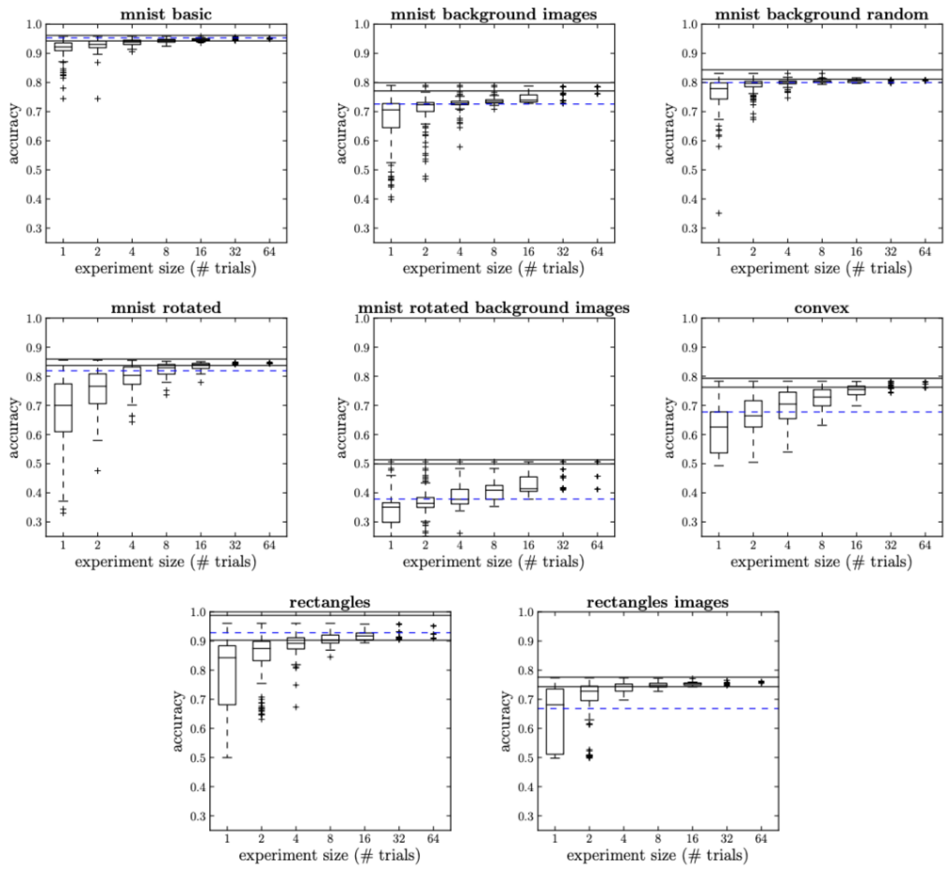


FIGURE 2.9 – Neural network performance when standard preprocessing algorithms are considered (9 hyper-parameters). Dashed blue line represents grid search accuracy using (on average) 100 trials , in which no preprocessing was done. From [24]

While these methods are simple to implement, they can be inefficient for high-dimensional search spaces and may not be able to effectively find the optimal hyperparameters.

It's important to note that these results are specific to the MNIST dataset and the specific deep neural network architecture used in the experiments. The performance of DENSER may vary for different datasets and models. However, the results on the MNIST dataset provide evidence of the potential of the DENSER method for hyperparameter optimization in deep neural networks.

2.3.3 Comparison with Bayesian optimization [25]

Bayesian optimization addresses the challenge of fine-tuning hyperparameters by building a probabilistic model of the relationship between the hyperparameters and the performance of the model. This allows the algorithm to make informed decisions about which hyperparameters to evaluate next, based on the expected improvement in performance.

In Bayesian optimization, the algorithm starts with a prior belief about the relationship between the hyperparameters and the performance of the model, and updates this belief with each evaluation of the model. Based on the updated belief, the algorithm selects the next set of hyperparameters to evaluate, with the goal of finding the optimal set of hyperparameters that maximize the performance of the model.

Bayesian optimization has been shown to be effective in a variety of hyperparameter tuning tasks and has been applied to a wide range of models, including neural networks, support vector machines, and gradient boosting models. The method can lead to faster convergence and better solutions compared to traditional hyperparameter tuning methods, especially for complex models with many hyperparameters.

In Bayesian optimisation, Gaussian processes are one of the preferred priors for quantifying the uncertainty in the objective function. However, estimating the hyper-parameters of the Gaussian process kernel with very few objective function evaluations is a daunting task, often with disastrous results as illustrated in [26] [27] [28] [29]

3 Research

3.1 Problematic

In recent years, evolutionary algorithms (EAs), deep learning (DL), and LSTM networks have become topics of great interest in data science. These modeling and prediction techniques have proven useful in a wide variety of applications, including time series prediction and animal movement tracking.

EAs are optimization methods that are based on principles inspired by biological evolution [30, 31]. In fact, EAs use techniques such as natural selection, mutation, and genetic recombination to create a population of candidate solutions and improve them over time. This technique has been used in optimize machine learning models, such as deep learning models, e.g., [32].

DL is a machine learning technique that uses deep neural networks to model and analyze data. These neural networks can have multiple layers and are trained using large amounts of data to improve their accuracy in prediction. DL has been used in object identification in images [33, 34], energy prediction [35] or fraud detection [36], among others. The computational cost of training in DL is typically high. For this reason transfer learning is gaining more and more popularity [37, 38, 39]. Transfer learning is a powerful machine learning technique that allows models to leverage knowledge learned from a source task to improve performance on a target task. In transfer learning, a pre-trained model is used as a starting point for a new task, rather than training a new model from scratch. This pre-trained model has typically been trained on a large, diverse dataset and has learned a set of general features that can be applied to many tasks.

Hyperparameter are parameters that are not learned directly from the data, but rather set by the researcher or programmer. Their optimization is a critical component of DL model development. In fact, such parameters affect the behavior of the model during training and can significantly impact the performance and generalization capability of the model. Therefore, selecting the optimal set of hyperparameters is crucial for achieving the best possible performance of the deep learning model. By optimizing such hyperparameters, DL models can achieve better accuracy and faster convergence, making them more suitable for real-world applications [40, 41]. In time series prediction, LSTM (Long Short-Term Memory) networks have become a popular technique. The reason being that such networks are capable of modeling temporal dependence in data and learning complex patterns in time series [42].

The above observations motivates us to introduce a neuroevolutionary transfer techniques in order to optimize LSTM networks. Our proposal is based on DENSER [21], which uses GAs and Dynamic Structured Grammatical Evolution in order to evolve DL networks. The original proposal is used to evolve CNN networks, while in this work we use a grammar for evolving LSTM networks. We use the neuroevolutionary technique to obtain an initial model for a domain , and then we fine tune the weights of the initial model on a target dataset.

3.2 Tools and Methodology

3.2.1 Environement

In this study, several tools were employed to aid in the development and analysis of the model.

TensorFlow and Keras were chosen as the primary deep learning frameworks for this project. These open-source libraries provide high-level APIs for building and training machine learning models. TensorFlow is a popular framework for developing and deploying machine learning models in production, while Keras is known for its user-friendliness and ease of use. By using both TensorFlow and Keras, the model development process was streamlined, and the resulting models were both accurate and efficient.

PyCharm and Jupyter Notebook were selected as the primary integrated development environments (IDEs) for coding the models. PyCharm is a professional-grade IDE that provides advanced features such as code completion, debugging, and version control integration. Jupyter Notebook, on the other hand, is a web-based notebook environment that allows for the creation and sharing of documents that contain live code, equations, visualizations, and narrative text. By using both PyCharm and Jupyter Notebook, the coding process was made more efficient and effective.

Anaconda was used as the main distribution platform for Python and its associated packages. It is an open-source distribution that simplifies package management and deployment. By using Anaconda, the installation and management of various packages, including TensorFlow, Keras, and Matplotlib, were made more efficient and streamlined.

Matplotlib, pandas, and numpy were used extensively for data visualization, analysis, and manipulation. Matplotlib is a popular library for creating static, animated, and interactive visualizations in Python, while pandas is a data manipulation library that provides data structures for efficiently storing and manipulating large datasets. Numpy, on the other hand, is a fundamental package for scientific computing that provides support for large, multi-dimensional arrays and matrices. By using these libraries, the analysis and interpretation of the data were made more efficient and effective.

Finally, two different types of computers were used for this research, one for development and the other for calculation. A development computer with high processing power and memory was used for coding and testing the models, while a calculation computer with even higher processing power and memory was used for the final training and evaluation of the models. This division of resources allowed for a more efficient use of the available computing power and ensured that the models were trained and evaluated in the most efficient manner possible.

3.2.2 Datasets

The data employed in this work was first proposed in [43], and consists of acceleration data collected on 20 Iberian pigs using a triaxial accelerometer. In particular the HOBO® Pendant G Data Logger (UA-004-64) [44] was used and it was physically located on the animals. This logger is capable of simultaneously recording acceleration and tilt by measuring an analogue signal in each of its three axes (X, Y and Z), and has 64 K bytes of data memory. The time required to fill the memory depends on how many channels are being recorded and the recording rate. The specific characteristics of the data collection process for this dataset are summarized in the following. As already mentioned, the data collection involved twenty pigs, and lasted twelve days. The animals were 5 months old and weighed approximately 40kg, and were located in an outdoor area of about 2500m2. The device was placed in the left ear of the pigs and set with a recording interval of 30 seconds. For more details about the dataset, please refer to [43].

The data described was processed in order to obtain a time series for each pig. The resulting time series will then be used as input to the forecasting algorithms that will be detailed later. First, we aggregate the three forces (the X, Y and Z axes) reads by the devices into a single signal called sum vector (S) :

$$S = \sqrt{X^2 + Y^2 + Z^2}$$

After this first phase, the data presented missing values, which could have been caused by failures in the devices, or by the time elapsed since the devices were placed in the different animals. Therefore, such gaps were filled using interpolation to try to minimize the impact on the final results [45] .Finally, the data were aggregated every 10 minutes. Aggregating time series values is a common practice in the literature [46, 47, 48]. It is usually employed to find more general results in the studies performed or to reduce large datasets.

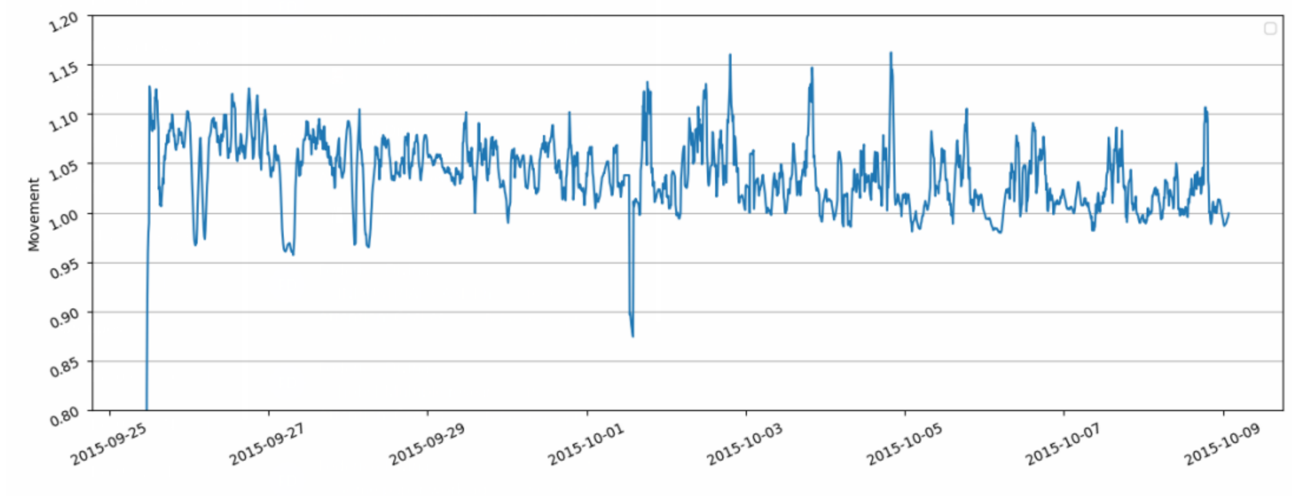


FIGURE 3.1 – Example of resulting time series, in particular for animal 2. In this figure the sum vector is shown

An example of resulting time series is shown in figure 3.1, where the activity value of animal 2 is reported over all the period of collection of the data. We can observe that in this period, the activity registered for this particular animal is usually within the range of 1 and 1.1, with some peaks that could indicate a very active or a very inactive state. In order to test whether or not the so obtained time series are stationary, we have analyzed the autocorrelation and the partial autocorrelation graphs of each time series. In figure 3.2 an example of such graphs, relative to animal 2, is shown. From this analysis, we concluded that all the graphs presented a few significant lags but these died out quickly, so we can conclude our series is stationary. In order to confirm this result, we have also applied the Augmented Dickey-Fuller Test to each time series [49]. The results confirm that all the time series are stationary. In order to use the time series we have transformed them as in [50, 51]. Each time series is transformed into a matrix, whose dimension depends on a historical window, w, and a prediction horizon, h, which are two parameters.

The historical window (w) represents the number of previous entries that are used in order to train an algorithm in order to obtain a model that will be used to predict the subsequent values (h). Figure 3.3 graphically shows this process.

In this work, the prediction horizon (h) has been set to 24, corresponding to a period of 4 hours, and the prediction horizon has been set to 128.

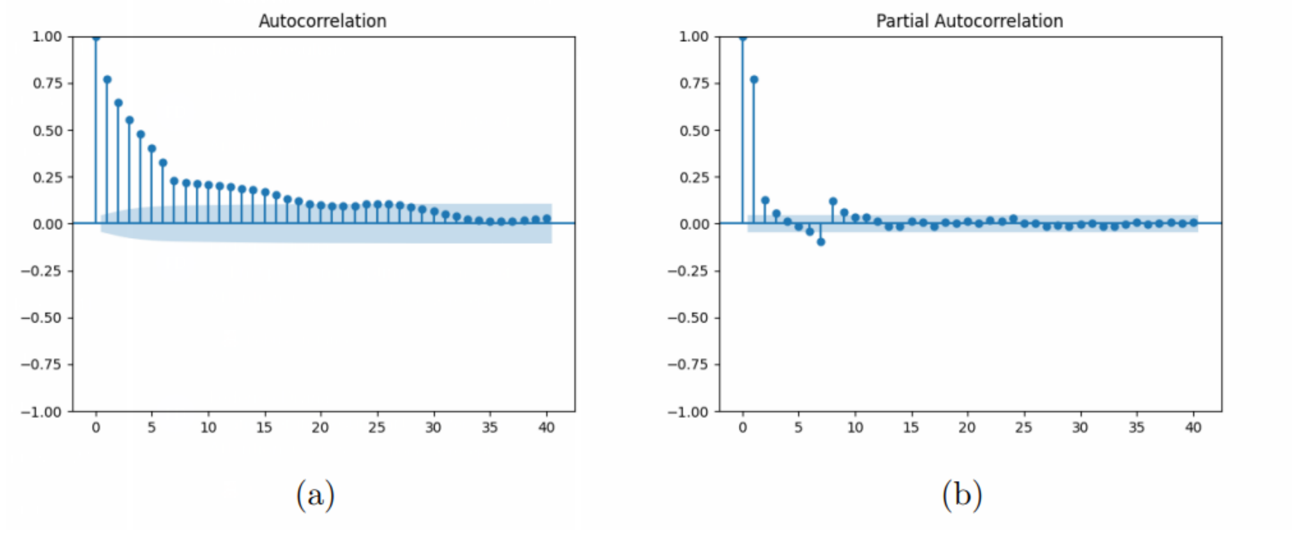


FIGURE 3.2 – Correlation plots for the time series relative to animal 2. (a) AutoCorrelation Function (ACF) ; (b) Partial AutoCorrelation Function (PACF)

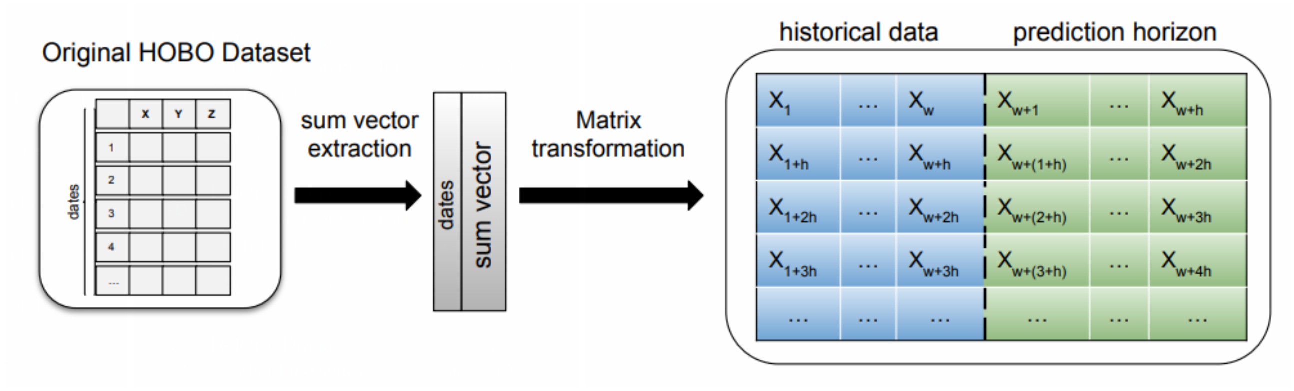


FIGURE 3.3 – Preprocessing of the data. w determines the amount of the historical data used, while h determines the prediction horizon. Xi represents a value of the sum vector.

3.3 Grammar

Our proposal consists in two phases. In the first phase, we use a neuroevolution technique, called DENSER, in order to obtain an initial model, using some data from the domain. In a second phase, this initial model is fine tuned on the final target datasets.

Deep Evolutionary Network Structured Representation (DENSER) [21] is a deep learning architecture that uses evolutionary optimization techniques to automatically discover and learn the optimal neural network structure and parameters for a given task by combining the basic principles of GAs with Dynamic Structured Grammatical Evolution (DSGE) [52]. The sequence of layers is encoded using the GA, and the parameterisation of each layer using DSGE.

In DENSER, the grammar is used to define the structure of the evolved networks. The grammar specifies a set of production rules that describe how the network can be built.

We used a grammar to generate a genotype describing the neural network architecture and hyperparameters, including the optimizer. We then generate the phenotype to generate a list of individuals representing our first generation. By using this approach, we can quickly and efficiently explore a wide range of neural network architectures and hyperparameters, enabling us to generate a population of individuals while keeping an easy to read and scalable grammar.

As said in 2.2.6, DSGE allow us to compute and generate the maximum number of derivations for each of the grammar's non-terminal symbols. In our case, the number of derivation define the number of LSTM layers used in a stacked LSTM neural network.

In our grammar, the architecture targeted is a single layered LSTM or a stacked LSTM network with a dropout layer to prevent overfitting by randomly dropping out or deactivating some of the neurons or connections in the network during training and finally a dense layer to provide a nonlinear mapping from the input to the output space, allowing the network to learn complex patterns and relationships in the data.

In addition of the architecture, our goal is also to discover the best set of hyperparameters in the different layers of the neural network.

In general, the grammar is written to describe the following structure :

1. hyperparameters selectionned throught the grammar :
   — units : In an LSTM (Long Short-Term Memory) network, the "units" hyperparameter refers to the number of memory cells or hidden units in each LSTM layer of the network. Each LSTM unit or cell contains a set of gates that regulate the flow of information into and out of the cell, allowing the network to selectively forget or remember previous input data.

     In general, increasing the number of units in an LSTM network can increase its capacity to capture more complex patterns in the input data, but may also require more training data and longer training times to avoid overfitting. The optimal number of units for a given task depends on the complexity of the problem and the amount of available data, and can be tuned through experimentation and cross-validation.

     In our definition of the grammar, the number of units is define as a random integer number between 1 and 20.

   — activation-function : In an LSTM (Long Short-Term Memory) network, the activation function hyperparameter refers to the nonlinear function that is applied to the output of each LSTM cell or hidden unit.

     The activation function is used to introduce nonlinearity into the network, allowing it to model complex relationships between the input and output data. Possible activation functions used in our grammar are the sigmoid function, the hyperbolic tangent (tanh) function, and the rectified linear unit (ReLU) function. A comparaison of the different activation function that can be used in an LSTM network can be found at [53]

- recurrent-function : The recurrent activation function acts as a gating mechanism that controls the flow of information from the previous time step to the current time step in the network. It takes as input the output of the previous LSTM cell and applies a nonlinear transformation to it before passing it as input to the current cell. Possible recurrent activation functions used in our grammar are the sigmoid function, the hyperbolic tangent (tanh) function, and the rectified linear unit (ReLU) function.
- bias : the bias hyperparameter refers to the scalar value added to the weighted sum of inputs and recurrent connections before applying the activation function in each LSTM cell or hidden unit.

  The bias term allows the LSTM network to learn an offset or bias in the input data that is not accounted for by the weights and the activation function. It helps the network to better model complex patterns and relationships in the input sequence.
- kernel-initializer : the kernel initializer hyperparameter refers to the method used to initialize the weights of the LSTM cells or hidden units.

  The kernel-initializer is responsible for determining the starting values of the weights of the LSTM cells, which are then updated during training through backpropagation. Initializing the weights properly can improve the performance and convergence of the network and help avoid problems such as vanishing or exploding gradients.
- recurrent-initializer : The recurrent-initializer is responsible for determining the starting values of the weights of the connections between the LSTM cells, which are then updated during training through backpropagation.
- bias-initializer : the bias initializer hyperparameter refers to the method used to initialize the biases of the LSTM cells or hidden units.

  The bias-initializer is responsible for determining the starting values of the biases of the LSTM cells, which are then updated during training through backpropagation.
- forget-bias : the forget-bias hyperparameter refers to a scalar value that is added to the forget gate bias during initialization.

  The forget gate is a key component of the LSTM cell that determines how much information from the previous cell state should be retained or forgotten. The forget gate bias is a parameter that determines how much the forget gate should be biased towards forgetting or retaining information from the previous cell state.

  By adding a forget-bias term during initialization, the network can be given a prior bias towards either remembering or forgetting information from the previous cell state.

2. dropout layer :
   - dropout rate : The dropout rate of a dropout layer in a neural network is a hyperparameter that determines the proportion of the input units (or neurons) that are randomly set to zero during each training iteration.

     The dropout technique is used to prevent overfitting in the neural network by randomly dropping out or disabling a fraction of the neurons in the layer during each training iteration. This can help prevent the network from becoming too specialized to the training data and improve its generalization ability to unseen data.

     The dropout rate is typically set to a value between 0.2 and 0.5, meaning that between 20% to 50% of the input units in the layer are randomly dropped out during training. The optimal value for the dropout rate depends on the specific problem and can be determined through experimentation and cross-validation.

3. dense layer :
   - The number of units is fixed at the same size as prediction horizon, see data preparation part
4. optimizer and macro :
   - optimizer :
     - gradient descent : Gradient descent optimizer is a popular algorithm used for training neural networks. It is an iterative optimization algorithm that works by minimizing the cost or loss function of the network by adjusting the values of the weights and biases of the network during the training process.

       The basic idea behind the gradient descent optimizer is to compute the gradient of the loss function with respect to the weights and biases of the network and update the values of the weights and biases in the opposite direction of the gradient, moving towards the minimum of the cost function. This process is repeated iteratively until the algorithm reaches a local minimum or convergence. [54]
     - rmsprop : RMSprop (short for Root Mean Square Propagation) is an extension of the gradient descent optimization algorithm that aims to reduce the impact of the vanishing and exploding gradient problem. It uses an adaptive learning rate that scales the gradients based on their historical magnitudes, allowing for faster convergence and more stable training.

       The RMSprop optimizer works by maintaining an exponential moving average of the squared gradients for each weight and bias in the network. This moving average is used to compute a scaling factor for the gradients during each iteration of the optimization process, allowing for the learning rate to be adaptively adjusted for each weight and bias.
     - Adam : Adam (Adaptive Moment Estimation) works by computing adaptive learning rates for each weight parameter based on both the first and second moments of the gradients. Specifically, it maintains an exponentially decaying average of the past gradients and their squares, which are used to compute the update for the weights during training.

       The key advantages of the Adam optimizer are its efficiency and robustness to noisy or sparse gradients, which makes it well-suited for large-scale deep learning applications. Additionally, it requires relatively little memory to store the parameter updates and can handle a wide range of different network architectures and learning rates.
   - macro :
     - batch size : batch size refers to the number of training examples used in one forward/backward pass of the network during a single iteration of the training algorithm. During each iteration of the training algorithm, the input data is divided into several smaller sets or "mini-batches," with each mini-batch consisting of a fixed number of examples.

       The batch size is a hyperparameter that is set before training the network and can have a significant impact on the training performance and convergence speed of the network.

$$\begin{aligned}
<start> ::= &<lstm> \;\mid\; <lstm><dropout>\\
<lstm> ::= &layer : lstm[units, int, 1, 1, 20]\\
&<activation-function><recurrent-function>\\
&<bias><kernel-initializer><recurrent-initializer>\\
&<forget-bias>\\
<dropout> ::= &layer : dropout[rate, float, 1, 0, 0.7]
\end{aligned}$$

FIGURE 3.4 – Grammar used for the encoding of LSTM and Dropout layers

A larger batch size can result in faster convergence due to the use of more training examples in each iteration, but it can also lead to higher memory requirements and longer training times. Conversely, a smaller batch size can lead to slower convergence but can require less memory and enable faster training.

First, for the grammar to be used in an ES, our goal is to provide a way to read and store a grammar file in a convenient format, which can later be used for initialization and decoding of individuals. A function as been implemented that performs two main tasks : reading the contents BNF grammar file and return it as a list of strings, where each string represents a line of the file, and then parsing the raw grammar into a dictionary object using the parseGrammar method. All the code for parsing and decoding the grammar is available at [55] under Apache-2.0 license.

Each solution encodes an ANN by means of an ordered sequence of feedforward layers and their respective parameters ; the learning, data augmentation, and any other hyperparameters can be encoded with each individual too. The representation of the candidate solution is made at two different levels :

— GA Level – encodes the macro structure of the networks and is responsible for representing the sequence of layers that later serves as an indicator to the grammatical starting symbol. It requires the definition of the allowed structure of the networks, i.e., the valid sequence of layers.

— DSGE Level – encodes the parameters associated to a layer. The parameters and their allowed values or ranges are codified in the grammar that must be defined by the user. Looking at the grammar of Figure 3.5 and 3.7. The same exercise can be made to the remaining layers defined in the grammar. The parameters can have closed values, or can assume a value in an integer or real interval.

$$\begin{aligned}
< bias >::=&bias : True \mid bias : False \\
< activation - function >::=& act : sigmoid \mid act : relu \mid act : softmax \\
&\mid act : softplus \mid act : softsign \mid act : tanh \\
&\mid act : selu \mid act : elu \\
< recurrent - function >::=&rec - act : sigmoid \mid rec - act : relu \\
&\mid rec - act : softmax \mid rec - act : softplus \\
&\mid rec - act : softsign \mid rec - act : tanh \\
&\mid rec - act : selu \\
&\mid rec - act : elu
\end{aligned}$$

FIGURE 3.5 – Grammar used for the activation and recurrent functions.

$$\begin{aligned}
< forget - bias >::=&f - bias : True \mid f - bias : False \\
< kernel - initializer >::=&kernel - init : glorot - uniform \\
&\mid kernel - init : glorot - normal \\
&\mid kernel - init : he - normal \\
&\mid kernel - init : he - uniform \\
&\mid kernel - init : random - normal \\
&\mid kernel - init : random - uniform \\
< recurrent - initializer >::=&recu - init : glorot - uniform \\
&\mid recu - init : glorot - normal \\
&\mid recu - init : he - normal \\
&\mid recu - init : he - uniform \\
&\mid recu - init : random - normal \\
&\mid recu - init : random - uniform
\end{aligned}$$

FIGURE 3.6 – Grammar used for the initializers of LSTM layers.

$$
\begin{aligned}
< learning >&::= < learner > [batchSize, int, 1, 16, 100] \\
< learner >&::= < gradient - descent > \mid < rmsprop > \mid < adam > \\
< gradient - descent >&::= learning : opt : gradient - descent[lr, float, 1, 0.0001, 0.1] \\
&\quad [momentum, float, 1, 0.68, 0.99][decay, float, 1, 0.000001, 0.001] \\
&\quad < nesterov > \\
< nesterov >&::= nesterov : True | nesterov : False \\
< adam >&::= learning : opt : adam[lr, float, 1, 0.0001, 0.1] \\
&\quad [beta1, float, 1, 0.5, 1][beta2, float, 1, 0.5, 1] \\
&\quad [decay, float, 1, 0.000001, 0.001] \\
< amsgrad >&::= amsgrad : True | amsgrad : False \\
< rmsprop >&::= learning : opt : rmsprop[lr, float, 1, 0.0001, 0.1] \\
&\quad [rho, float, 1, 0.5, 1][decay, float, 1, 0.000001, 0.001]
\end{aligned}
$$

Figure 3.7 – Grammar used for the optimizers

3.4 Neuroevolution

In the process of population creation, we leverage the power of DSGE to produce a diverse and dynamic set of genotypes. These genotypes serve as the foundation upon which the individuals that undergo evolution will be constructed. The utilization of grammar allows for a structured and systematic approach to generating these genotypes, ensuring that they are representative of the range of possible variations within the population. The evolutionary engine is a $(1 + \lambda) - ES$: the next generation is formed by the best individual (elite), and λ mutations of it.

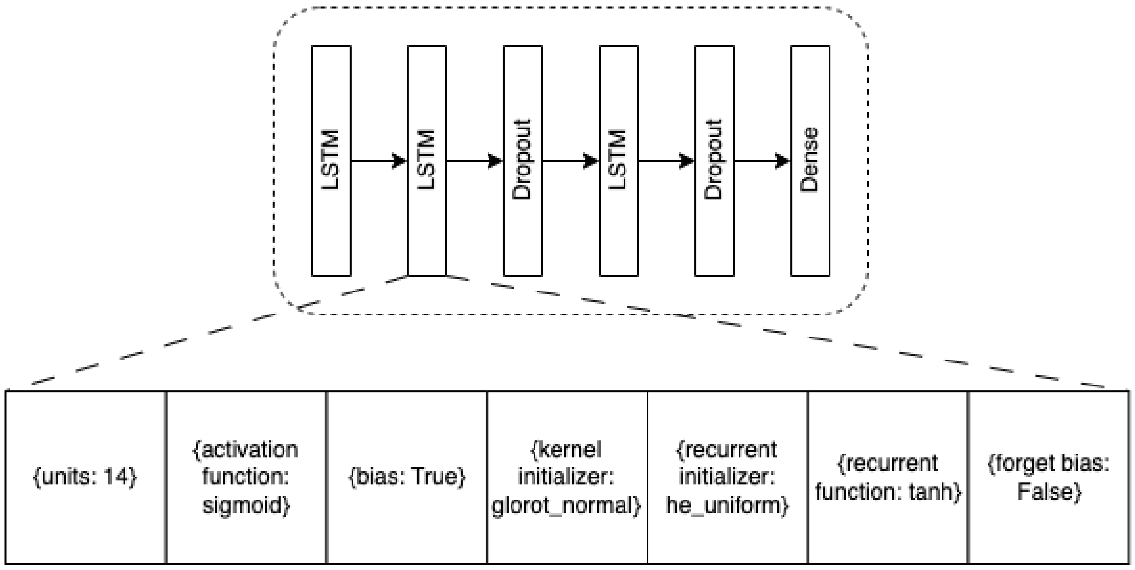

LSTM
LSTM
Dropout
LSTM
Dropout
Dense
{units: 14}
{activation function: sigmoid}
{bias: True}
{kernel initializer: glorot_normal}
{recurrent initializer: he_uniform}
{recurrent function: tanh}
{forget bias: False}

FIGURE 3.8 – Example of genotype of a candidate solution that encodes a LSTM layer.

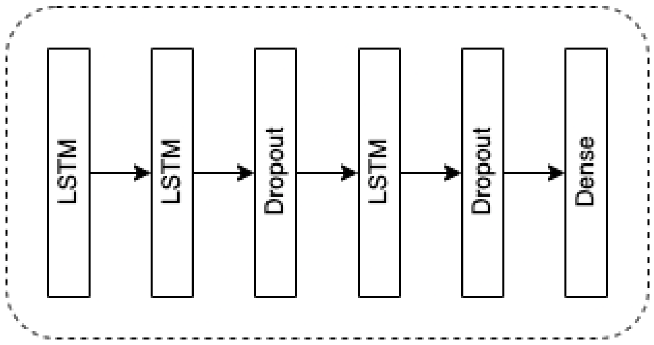


FIGURE 3.9 – Example of candidate solution macro structure described by the grammar

To evolve our population, we use variation operators, that play a crucial role in evolutionary algorithms such as ES and GA. These operators are responsible for generating new candidate solutions by applying genetic operations to the existing population. This section will explore the various types of variation operators employed, focusing on their characteristics and functions.

3.4.1 Crossover

In this work, a module is defined as a collection of layers that share the same GA structure index, rather than a group of layers that can be replicated multiple times. To facilitate layer exchanges within a module, a uniform crossover technique called bit-mask crossover is employed.

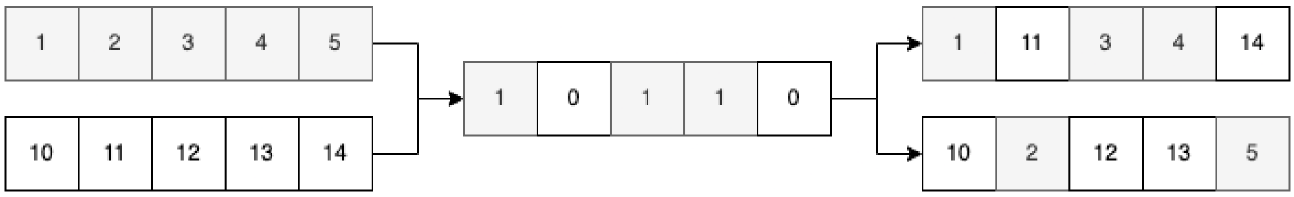


FIGURE 3.10 – bit-mask crossover example

This involves creating a matrix of randomly selected 0s and 1s with the same length as the number of codons. To generate the first offspring, a codon is copied from the first parent if the corresponding bit in the matrix is 1, and from the second parent if it is 0. The process is reversed for the second offspring.

3.4.2 Mutation

The first mutation operator used is the modification of the selected gene by applying a small random change where an integer/float block is replaced by a new one. For integers we generate new integers at random ; for floats a Gaussian perturbation is used.

We also introduced a new mutation operator that does not change any of the layer structure and/or learning parameters, and increases the train time of the individual. Whilst in F-DENSER the maximum train time is set the same for all individuals, in FDENSER++ [22] the maximum train time is set independently for each individual : in the initial population all individuals are trained for the same amount of time, and the mutation operator changes the maximum train time ; any of the other mutation operators reset the evaluation time to the default value, so that the offspring solutions are not evaluated for longer than necessary. Therefore, in our proposition, each selected individual as a probability to add 50 epochs to his training time.

3.4.3 Evaluation

The dataset to be used in the evolution process is selected randomly from among the 20 pre-processed datasets available. Once the initial generation of individuals is created, each individual undergoes 500 epochs of training, utilizing the Mean Squared Error (MSE) (1) measurement as the loss function.

To determine the best individual across generations, the "evaluate" method of Keras [12] has been utilized with the MSE score as a measurement. Additionally, elitism has been applied to ensure that the best models are preserved throughout the generations.

3.5 Transfer Learning

To overcome the challenge of high computational cost in our deep learning approach, we have incorporated transfer learning, which is widely accepted as a highly effective technique in the field. This technique has been shown to deliver remarkable performance across diverse domains, as evidenced in recent studies such as [56, 57, 58].

The key idea behind transfer learning is to leverage pre-trained models that have already been developed on a reference dataset and then use them as a starting point for learning on new datasets. This approach is especially valuable when the new datasets have limited amounts of labeled data or when training deep learning models from scratch on such datasets would be prohibitively expensive in terms of time and computational resources.

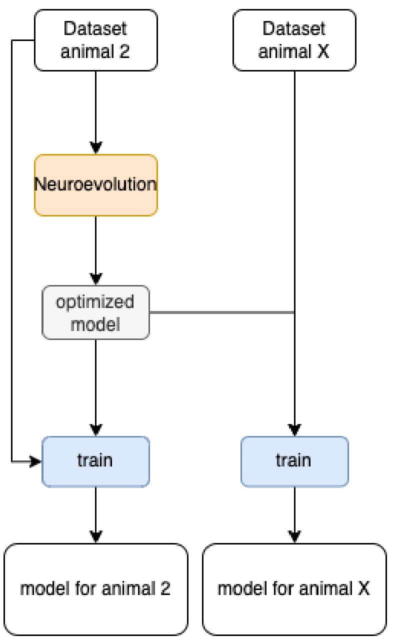


FIGURE 3.11 – Transfer learning method used to generate the model of each animal

To apply this technique in our work, we first obtain an initial model by training it on a reference dataset that is relevant to our problem domain. We then use this model as the starting point for learning on our final datasets, which are specific to our research objectives. By further training the model for an additional 500 epochs on these final datasets, we enable it to learn more sophisticated and nuanced representations of the underlying patterns and relationships in the data.

Overall, this approach allows us to effectively leverage the knowledge and insights gained from pre-trained models and tailor them to the specific needs of our research, thus enabling us to achieve high-quality results with less time and resources.

As the initial model after the evolutionary process is underfited, we also train it for 500 more epochs.

3.6 Application of the methodology

Firstly, we ran the Evolutionary Strategy (ES) for 20 generations with a population size of 20, utilizing the DENSER selection strategy. The crossover probability was set to 0.1, while the mutation probability was set to 0.3. Each individual was trained for 500 epochs, and the grammar detailed in section 3.3 was utilized with a maximum recursion level of 2.

At the conclusion of the ES, the best individual, based on the Mean Squared Error (MSE), was selected and used as the initial model in the second phase of our proposed technique. The selected initial model was then further trained for an additional 500 epochs on the final dataset. It is important to note that the data used to obtain the initial model was randomly selected from the available data related to animal 2.

The best architecture discovered during the initial network creation phase was a two-layered stacked LSTM neural network. The first layer was composed of 13 units and utilized a sigmoid activation function with a hyperbolic tangent recurrent activation function. The kernel and recurrent weights of the first layer were initialized using He normal and Glorot uniform initializers, respectively. Additionally, the forget bias was set to false.

The second layer of the stacked LSTM network was similar to the first layer, but with the addition of a bias term. The kernel and recurrent weights of the second layer were initialized using random normal and random uniform initializers, respectively. The dropout rate of the Dropout layer was approximately 0.015039, as determined by the Genetic Algorithm (GA). A visual representation of the topology can be found in Fig. 3.12

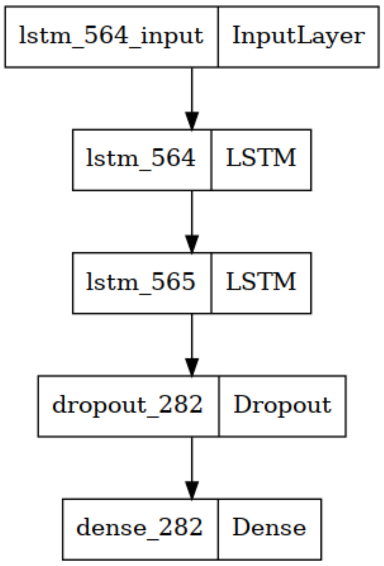


FIGURE 3.12 – Topology of the RNN that reports the best results

Overall, the process we followed to obtain the initial network for our proposed technique was carefully designed and executed with precision. The use of the ES allowed for the exploration of a wide range of possible architectures, while the utilization of a grammar-based approach ensured the structure of the network was consistent and well-formed. The resulting network was well-suited to the task at hand and served as a strong foundation for the subsequent phases of our proposed technique.

Table 1, reports the results obtained on animal 2, which was used for obtaining the initial model, the best and worst results achieved (on animal 17 and animal 1, respectively), and the average results on the 20 time series considered. Predictions made on animal 2 are shown on Fig. 3.13 Je ne sais pas quoi ecrire donc je tape n'importe quoi pour voir comment ca ce termine

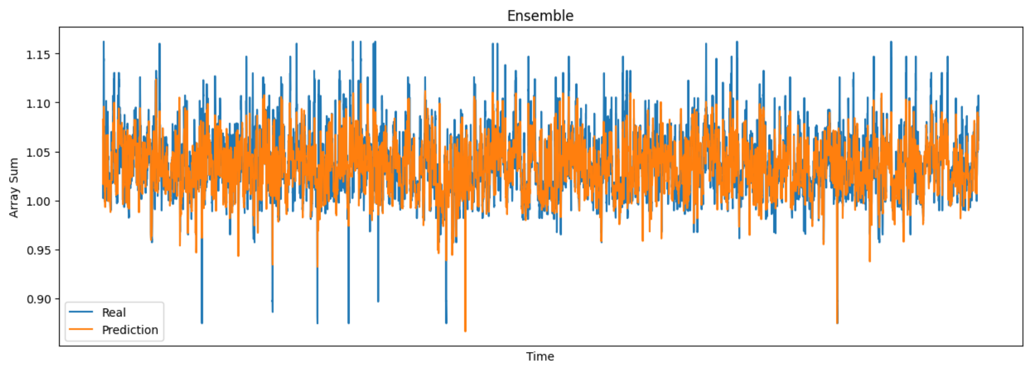
We can notice that the initial model transferred quite well on all the cases, and in some cases, like on animal 17, it even achieve better results than for the animal on which the initial model was obtained. Both best and worst prediction are shown in Fig. 3.14

The different metrics used to estimate the performance of the model are presented as follow :

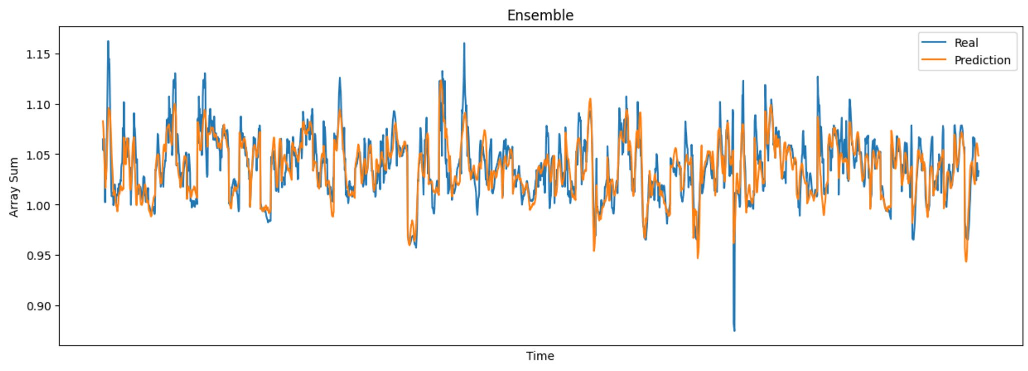
— MAPE : The Mean Absolute Percentage Error measures the average absolute percentage difference between the predicted values and the actual values. The lower the MAPE, the better the accuracy of the predictions. In this case, the average MAPE value of 0.01224 indicates that the proposed technique has an accuracy of about 98.77% on average.

— MAE : The Mean Absolute Error measures the average absolute difference between the predicted values and the actual values. A lower value of MAE indicates better accuracy. The average MAE value of 0.01264 suggests that the proposed technique is able to predict the output values with an average error of 0.01264.

— MSE : The Mean Squared Error measures the average of the squared differences between the predicted and actual values. The lower the MSE, the better the accuracy of the predictions. The average MSE value of 0.00031 indicates that the proposed technique has a low average squared error.



(a) All animal 2



(b) Subset animal 2

FIGURE 3.13 – Example of resulting time series prediction achieved by the model on animal 2.

— MRE : The Mean Relative Error measures the average percentage difference between the predicted values and the actual values. A lower MRE indicates better accuracy. The average MRE value of 1.22479 indicates that the proposed technique's predictions are on average within 22.48% of the actual values.

$$MSE : (y, \hat{y}) = \frac{\sum_{i=0}^{N-1} (y_i - \hat{y}_i)^2}{N} \quad (1)$$

$$MAPE : (y, \hat{y}) = \frac{100\%}{N} \sum_{i=0}^{N-1} \frac{y_i - \hat{y}_i}{y_i} \quad (3)$$

$$MAE : (y, \hat{y}) = \frac{\sum_{i=0}^{N-1} |y_i - \hat{y}_i|}{N} \quad (2)$$

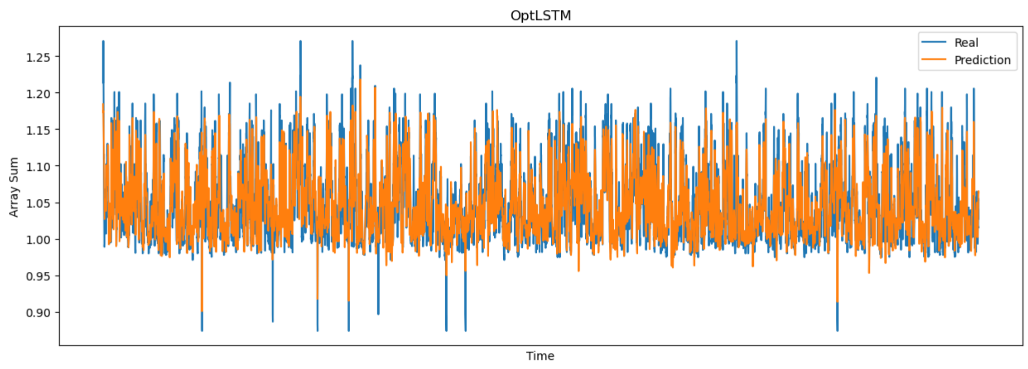
$$MRE : (y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N-1} \frac{|y_i - \hat{y}_i|}{|y_i|} \quad (4)$$

Upon examining the evaluations of the models trained with the data from the 20 different datasets, it can be concluded that the results obtained are quite satisfactory, as can be seen from the last row of the Table.

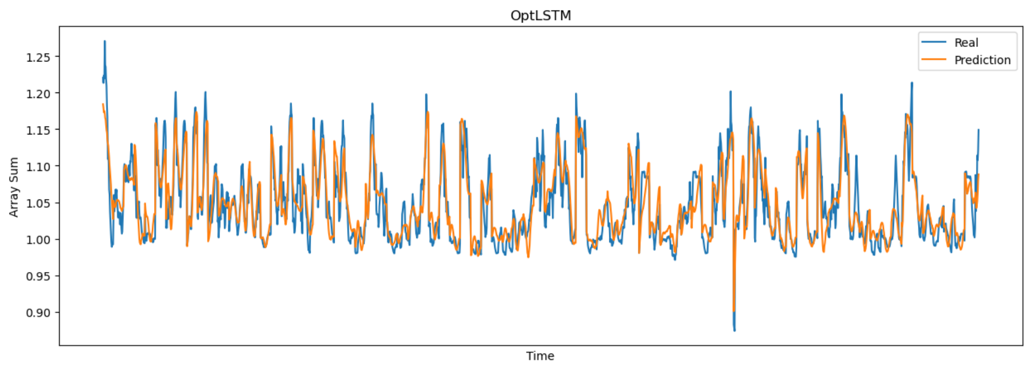
To globally evaluate the performance of the models, we have performed a comparisons with other commonly used methods for time series prediction, i.e, Gaussian Process [59], Gradient Boosting [60], Random Forest [61] and LightGBM [62].

	MAPE	MAE	MSE	MRE
animal 1	0.01680	0.01771	0.00054	1.68071
animal 2	0.01205	0.01253	0.00029	1.20500
animal 3	0.01181	0.01196	0.00026	1.18131
animal 4	0.01179	0.01281	0.00034	1.17991
animal 5	0.01082	0.01078	0.00022	1.08235
animal 6	0.01081	0.01119	0.00023	1.08164
animal 7	0.01338	0.01375	0.00034	1.33895
animal 8	0.01101	0.01141	0.00025	1.10149
animal 9	0.01233	0.01261	0.00030	1.23390
animal 10	0.01103	0.01111	0.00024	1.10363
animal 11	0.01103	0.01127	0.00024	1.10310
animal 12	0.01624	0.01662	0.00049	1.62499
animal 13	0.01152	0.01186	0.00027	1.15257
animal 14	0.01268	0.01282	0.00031	1.26800
animal 15	0.01646	0.01688	0.00054	1.64692
animal 16	0.01255	0.01313	0.00030	1.25523
animal 17	0.00688	0.00712	0.00008	0.68875
animal 18	0.00816	0.00871	0.00013	0.81653
animal 19	0.01379	0.01434	0.00037	1.37926
animal 20	0.01382	0.01437	0.00038	1.38239
average	0.01224	0.01264	0.00031	1.22479

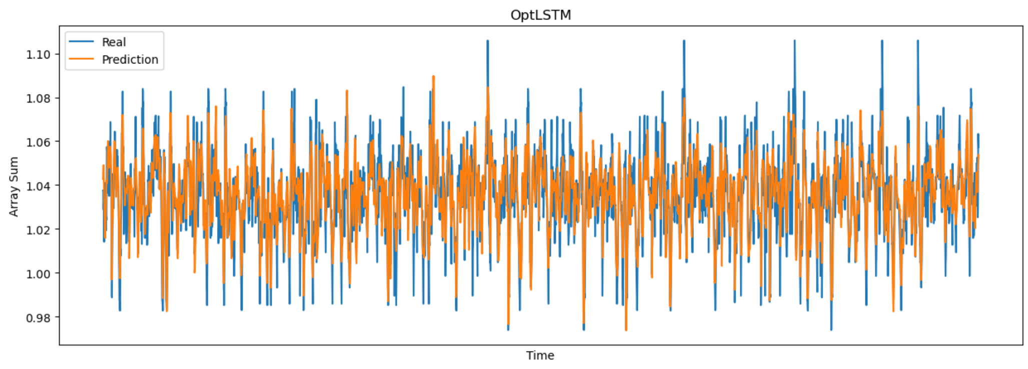
Table 1 – Results obtained on test data for all animals, and average results.



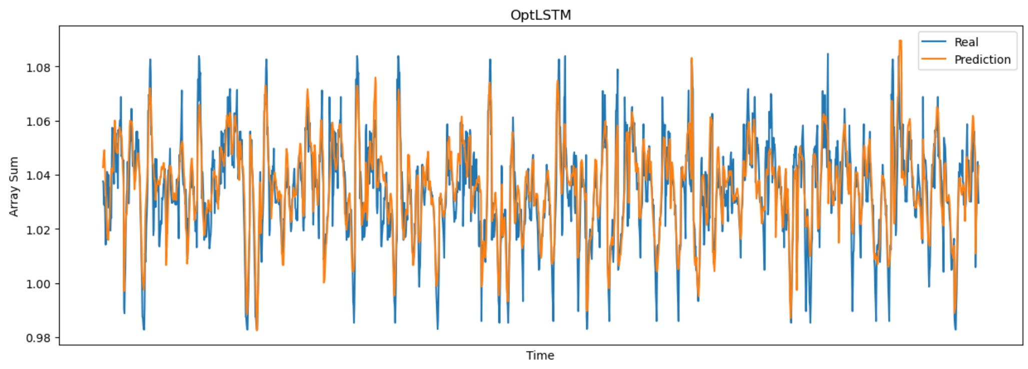
(a) All animal 1



(b) Subset animal 1



(c) All animal 17



(d) Subset animal 17

FIGURE 3.14 – Examples of best and worst predictions vs real values. Best predictions were achieved on animal 17, worst on animal 1.

3.7 Comparison

3.7.1 Gaussian process :

Gaussian processes (GPs) [59] are a popular statistical modeling technique used for time series prediction. They are a non-parametric approach to modeling complex data distributions, where the distribution over functions is modeled directly. GPs are useful for time series prediction because they can provide a measure of uncertainty along with a point estimate, which can be used to make informed decisions.

In a GP, the prior distribution over functions is assumed to be a multivariate normal distribution, where the mean function and covariance function are specified by the user. The mean function represents the expected value of the function at each point in time, while the covariance function specifies the degree of similarity between different time points.

During the training phase, the GP model is fitted to the observed data by updating the prior distribution using the likelihood function. The likelihood function specifies the probability of observing the data given the model parameters, which are the mean function and covariance function.

Once the model is trained, it can be used to make predictions by computing the posterior distribution over functions given the observed data. This is done using Bayesian inference, where the posterior distribution is proportional to the product of the likelihood and the prior distribution.

The advantage of using GPs for time series prediction is that they can capture complex patterns in the data without making strong assumptions about the underlying data distribution. They can also be used to estimate uncertainty in the predictions, which is essential for decision-making in many real-world applications. However, GPs can be computationally expensive and require careful selection of hyperparameters to achieve optimal performance.

Overall, Gaussian processes are a powerful technique for time series prediction that can provide both point estimates and a measure of uncertainty. They have been used successfully in a wide range of applications, including finance, climate modeling, and medical monitoring. As more data becomes available and computing power increases, it is likely that GPs will continue to be a popular approach for time series prediction.

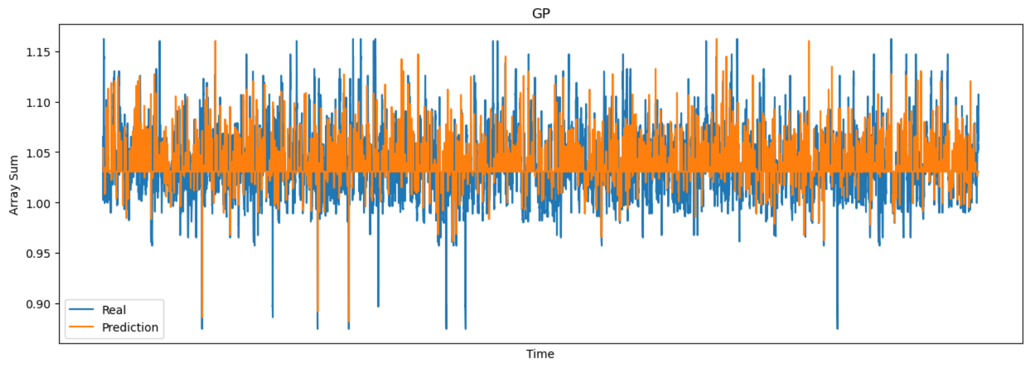
The GP in this work is configured with the following parameters :

- Population size : 600
- Generations : 20
- Stopping criteria : 0.001
- Function set : ['add', 'sub', 'mul', 'div', 'sqrt', 'log', 'abs', 'neg', 'inv', 'max', 'min']
- Metric : Mean squared Error
- Crossover probability : 0.7
- Subtree mutation probability : 0.1
- Hoist mutation probability : 0.05
- Point mutation probability : 0.1
- Maximum samples : 0.9 (fraction of samples used for training)
- Parsimony coefficient : 0.01

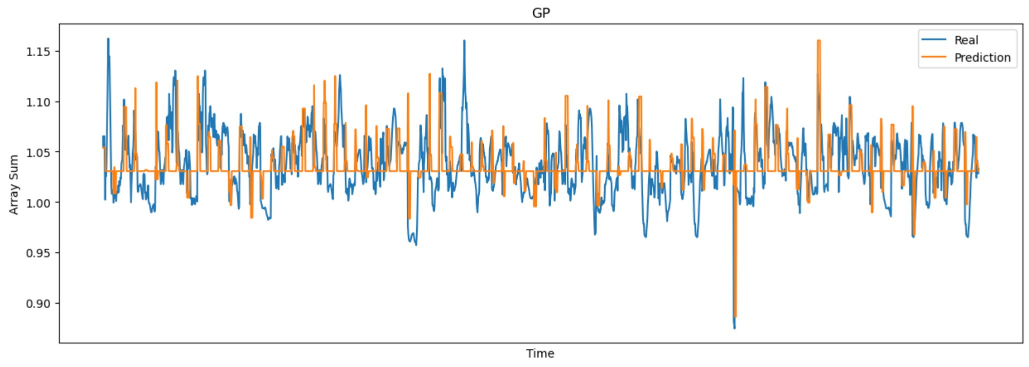
These parameters are used to configure the GP regressor for fitting and optimizing a symbolic regression model.

	MAPE	MAE	MSE	MRE
animal 1	0.03459	0.03669	0.00266	3.45968
animal 2	0.02447	0.02540	0.00110	2.44730
animal 3	0.02336	0.02361	0.00105	2.33679
animal 4	0.03019	0.03257	0.00203	3.01993
animal 5	0.01891	0.01885	0.00072	1.89136
animal 6	0.02314	0.02398	0.00110	2.31446
animal 7	0.02691	0.02758	0.00146	2.69187
animal 8	0.02200	0.02290	0.00103	2.20030
animal 9	0.02706	0.02777	0.00172	2.70641
animal 10	0.02330	0.02353	0.00115	2.33068
animal 11	0.02340	0.02390	0.00114	2.34069
animal 12	0.02831	0.02925	0.00182	2.83140
animal 13	0.02202	0.02275	0.00105	2.20256
animal 14	0.02458	0.02500	0.00117	2.45837
animal 15	0.02802	0.02867	0.00167	2.80232
animal 16	0.03038	0.03173	0.00205	3.03858
animal 17	0.01851	0.01912	0.00060	1.85129
animal 18	0.01768	0.01887	0.00055	1.76868
animal 19	0.02469	0.02558	0.00125	2.46978
animal 20	0.02498	0.01349	0.00163	2.49807
average	0.02483	0.02569	0.00135	2.48303

Table 2 – Results obtained on test data for all animals with the gaussian process method, and average results.



(a) All animal 1



(b) Subset animal 1

FIGURE 3.15 – Examples of predictions vs real values with the gradient boosting method. Prediction are made on animal 1.

3.7.2 Gradient boosting :

Gradient boosting is a machine learning technique Gradient Boosting [60] used for making predictions based on historical data. In the context of time series prediction, gradient boosting involves training a series of decision tree models on past data and then combining them to form a powerful ensemble model.

The process begins by fitting a base decision tree model to the historical data. This model is typically shallow and simple, allowing it to quickly capture any linear relationships in the data. The residuals or errors of the first model are then calculated and used to train a second decision tree model. This second model is trained to predict the residuals of the first model, rather than the original target variable. This process of fitting a decision tree model to the residuals is repeated several times until the desired level of accuracy is achieved.

The final model is an ensemble of all the individual decision trees. When making a prediction, the ensemble model combines the predictions of each individual tree, weighted by their performance on the training data. This allows the model to capture both linear and nonlinear relationships in the data, making it highly effective for time series prediction.

One of the key advantages of gradient boosting is its ability to handle missing data and outliers. The decision tree models used in the process are able to work with incomplete data and can effectively handle outliers by assigning them a high residual value. Additionally, the process of fitting decision trees to the residuals can help to reduce the impact of noise in the data.

Gradient boosting has been successfully applied to a wide range of time series prediction problems, including financial forecasting, stock price prediction, and weather forecasting. It is a highly flexible and powerful technique that can be adapted to different types of data and modeling requirements. However, like any machine learning technique, it requires careful tuning and parameter selection to achieve optimal performance.

The GB model is configured with the following parameters :

— Number of estimators : 100

— Maximum depth : 4

	MAPE	MAE	MSE	MRE
animal 1	0.01784	0.01890	0.00067	1.78460
animal 2	0.01431	0.01484	0.00043	1.43172
animal 3	0.01269	0.01286	0.00030	1.26930
animal 4	0.01422	0.01537	0.00046	1.42262
animal 5	0.01077	0.01076	0.00023	1.07700
animal 6	0.01221	0.01263	0.00029	1.22106
animal 7	0.01458	0.01498	0.00040	1.45862
animal 8	0.01143	0.01186	0.00027	1.14337
animal 9	0.01513	0.01553	0.00052	1.51389
animal 10	0.01160	0.01169	0.00028	1.16014
animal 11	0.01220	0.01247	0.00029	1.22066
animal 12	0.01511	0.01559	0.00047	1.51194
animal 13	0.01168	0.01200	0.00029	1.16864
animal 14	0.01314	0.01331	0.00033	1.31415
animal 15	0.01493	0.01527	0.00044	1.49382
animal 16	0.01750	0.01835	0.00062	1.75099
animal 17	0.00898	0.00929	0.00016	0.89856
animal 18	0.00854	0.00913	0.00014	0.85416
animal 19	0.01382	0.01438	0.00039	1.38274
animal 20	0.01419	0.01480	0.00045	1.41988
average	0.01324	0.01370	0.00037	1.32489

Table 3 – Results obtained on test data for all animals with the gradient boosting method, and average results.

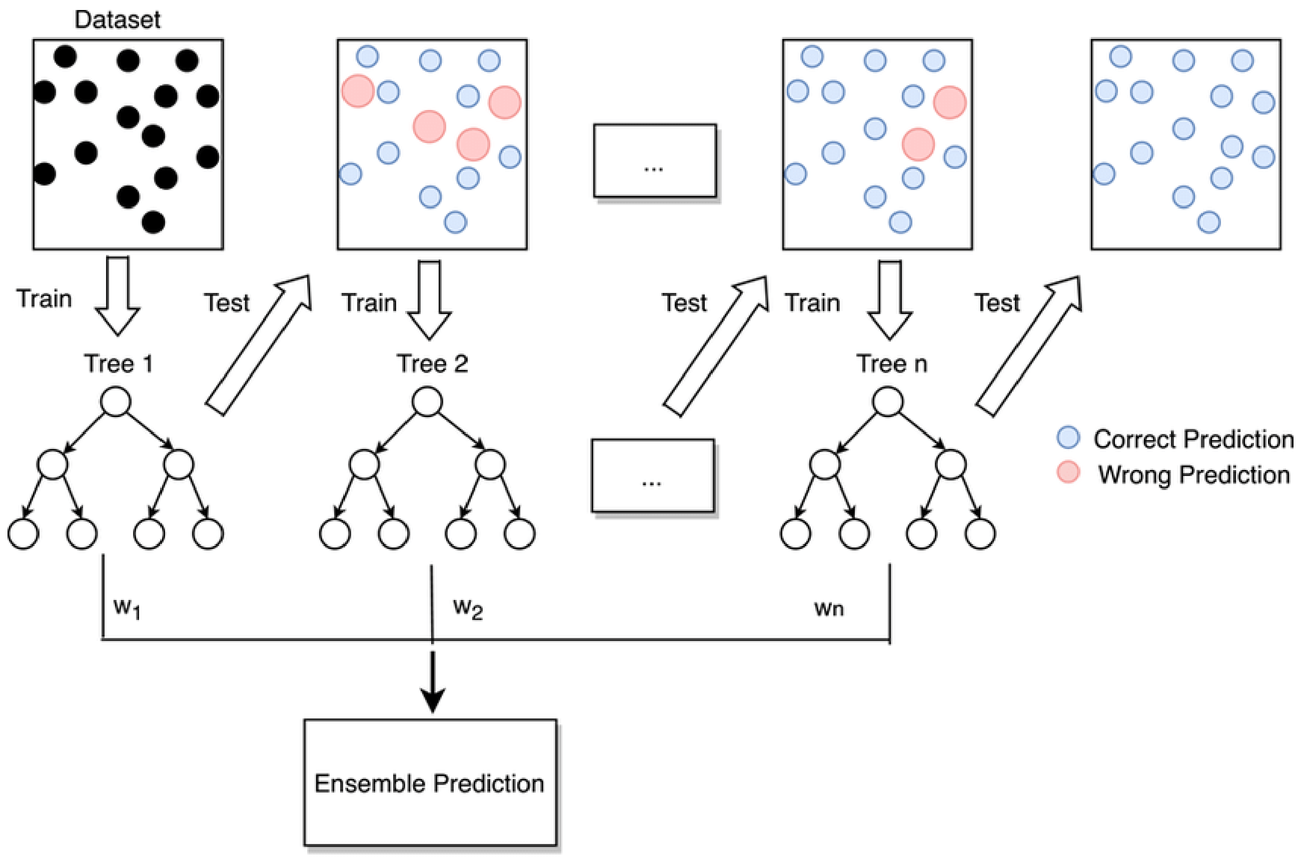
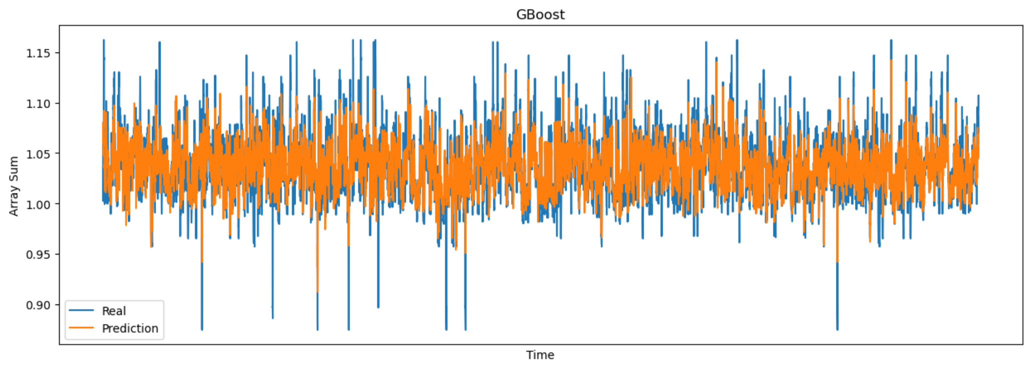
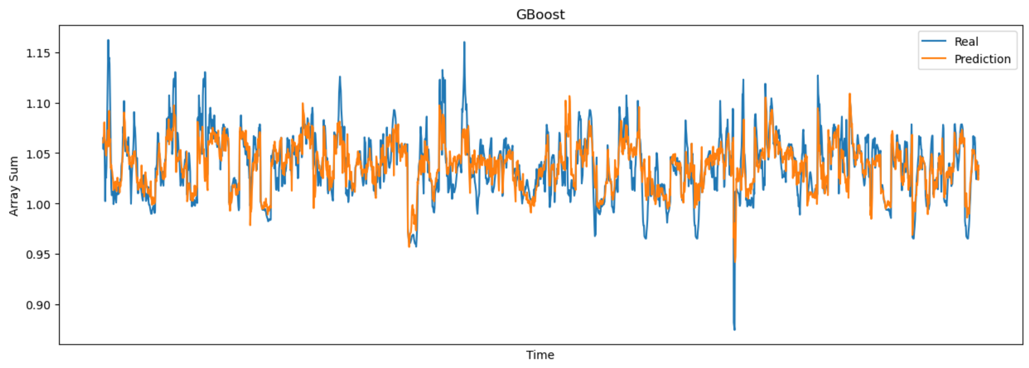


FIGURE 3.16 – Flow diagram of gradient boosting machine learning method, from [63]



(a) All animal 1



(b) Subset animal 1

FIGURE 3.17 – Examples of predictions vs real values with the gradient boosting method. Prediction are made on animal 1.

3.7.3 Random Forest :

Random forests Random Forest [61] have become a popular machine learning algorithm for time series prediction due to their ability to handle high-dimensional data, capture non-linear relationships, and provide robustness against overfitting. Random forests are an ensemble learning method that combines multiple decision trees to create a powerful predictive model.

The process of building a random forest model for time series prediction involves the following steps :

Data preparation : As with other machine learning algorithms, the first step in building a random forest model is to prepare the data. This involves cleaning the data, handling missing values, and transforming the data into a format suitable for machine learning. For time series data, it is also essential to split the data into training and testing sets.

Feature selection : Random forests have the ability to handle a large number of features, but not all features may be relevant for predicting the target variable. Feature selection techniques such as correlation analysis or recursive feature elimination can be used to identify the most important features for the model.

Training the random forest model : Once the data is prepared and the relevant features are selected, the random forest model can be trained. The model is built by creating multiple decision trees, each of which is trained on a random subset of the features and a random subset of the training data. The final prediction of the random forest model is the average prediction of all decision trees.

Evaluating the model : After training the random forest model, it is essential to evaluate its performance on the testing set. Common metrics used for evaluating the performance of a random forest model include mean squared error (MSE), mean absolute error (MAE), and correlation coefficient.

Tuning the hyperparameters : Hyperparameters of the random forest model such as the number of trees, the maximum depth of the tree, and the minimum number of samples required to split a node can be tuned to improve the performance of the model.

Overall, random forests provide a robust and flexible approach to time series prediction. They can handle a variety of data types and sizes, are resistant to overfitting, and can be easily tuned to improve performance. As a result, they have become a popular choice for many time series prediction tasks.

The Random Forest model is configured with the following parameters :

- Number of estimators : 500
- Maximum depth : 8
- Minimum samples required to split an internal node : 5
- Learning rate : 0.01
- Loss function : least squares regression

	MAPE	MAE	MSE	MRE
animal 1	0.01900	0.02018	0.00079	1.90042
animal 2	0.01479	0.01536	0.00046	1.47965
animal 3	0.01335	0.01352	0.00032	1.33513
animal 4	0.01557	0.01682	0.00055	1.55754
animal 5	0.01107	0.01106	0.00025	1.10716
animal 6	0.01255	0.01295	0.00032	1.25508
animal 7	0.01480	0.01520	0.00042	1.48013
animal 8	0.01203	0.01249	0.00031	1.20379
animal 9	0.01506	0.01546	0.00053	1.50609
animal 10	0.01250	0.01262	0.00032	1.25073
animal 11	0.01287	0.01316	0.00033	1.28738
animal 12	0.01566	0.01621	0.00055	1.56625
animal 13	0.01191	0.01224	0.00032	1.19180
animal 14	0.01388	0.01408	0.00039	1.38851
animal 15	0.01551	0.01587	0.00047	1.55156
animal 16	0.01700	0.01780	0.00059	1.70044
animal 17	0.01022	0.01058	0.00020	1.02292
animal 18	0.00981	0.01048	0.00017	0.98105
animal 19	0.01410	0.01464	0.00043	1.41043
animal 20	0.01499	0.01566	0.00050	1.49972
average	0.01383	0.01423	0.00041	1.38379

TABLE 4 – Results obtained on test data for all animals with the random forest method, and average results.

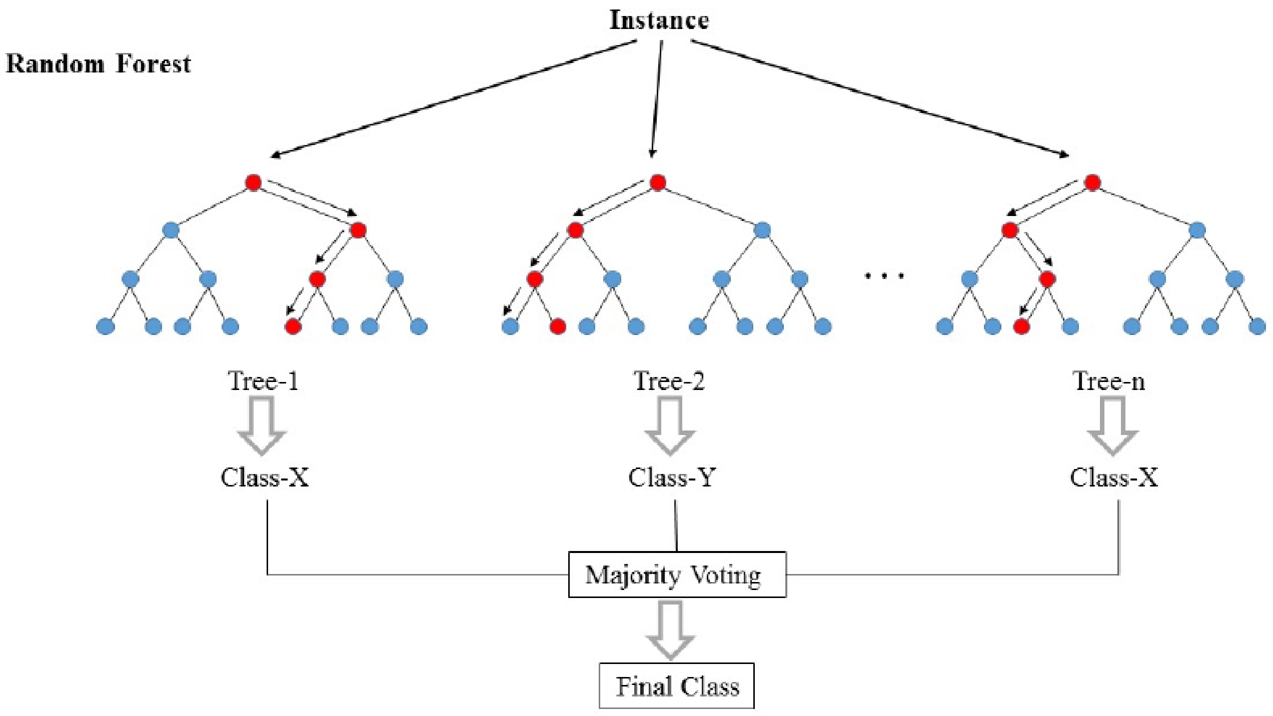
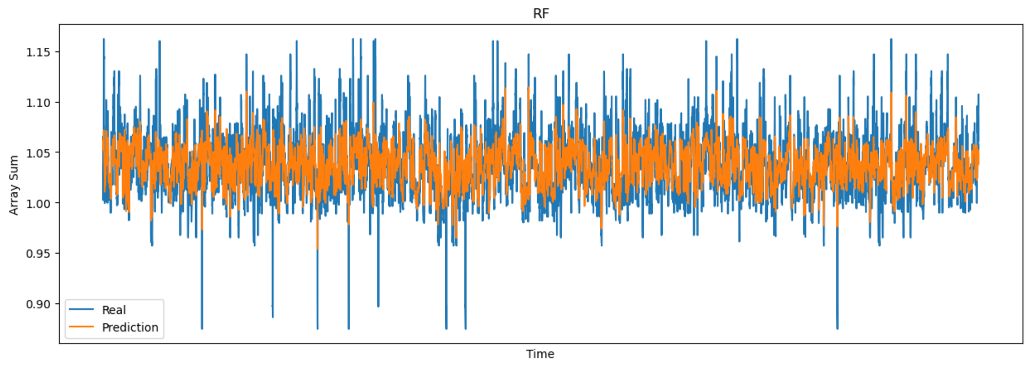
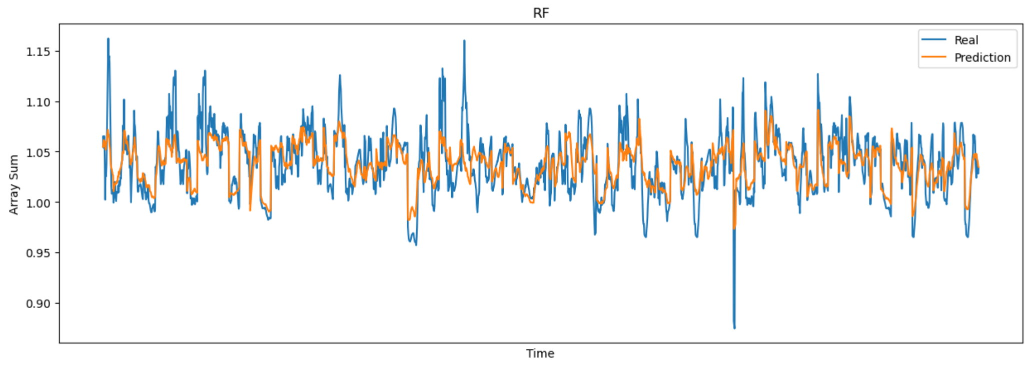


FIGURE 3.18 – Flow diagram of random forest algorithm



(a) All animal 1



(b) Subset animal 1

FIGURE 3.19 – Examples of predictions vs real values with the random forest method. Prediction are made on animal 1.

3.7.4 Light GBM :

LightGBM (Light Gradient Boosting Machine) LightGB [62] is a machine learning algorithm that uses gradient boosting to perform fast and accurate predictions on large-scale datasets. LightGBM is a relatively new algorithm, but it has quickly gained popularity due to its exceptional performance, scalability, and flexibility.

LightGBM is particularly well-suited for time series prediction tasks because it can handle both structured and unstructured data, making it suitable for various data types commonly found in time series datasets. It also offers support for missing values, categorical features, and user-defined objective functions, which can be customized for specific use cases.

The algorithm works by creating a series of decision trees, where each tree builds upon the errors of the previous tree to improve the overall prediction accuracy. The key innovation of LightGBM is its ability to split data points by feature values, rather than by feature presence, as in traditional decision trees. This approach reduces memory consumption and speeds up training while maintaining or even improving accuracy.

LightGBM also features a unique algorithm for parallel learning, which allows it to process large-scale datasets efficiently. The algorithm can automatically detect the best parallel learning strategy based on the available resources, reducing the need for manual configuration.

Overall, LightGBM is a powerful tool for time series prediction tasks, with its speed, accuracy, and flexibility making it an excellent choice for researchers and practitioners alike.

The LightGBM model is configured with the following parameters :

— Number of leaves : 31

— Learning rate : 0.05

— Number of estimators : 200

— Objective : regression

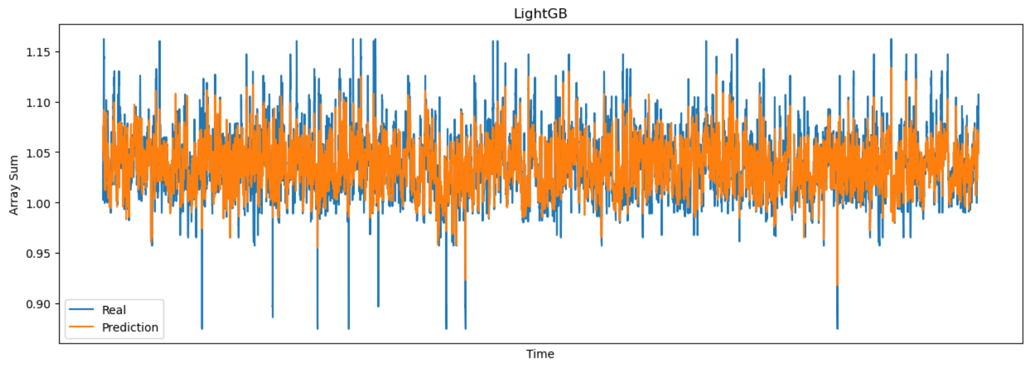
— Linear tree : True

3.7.5 Non Optimized LSTM

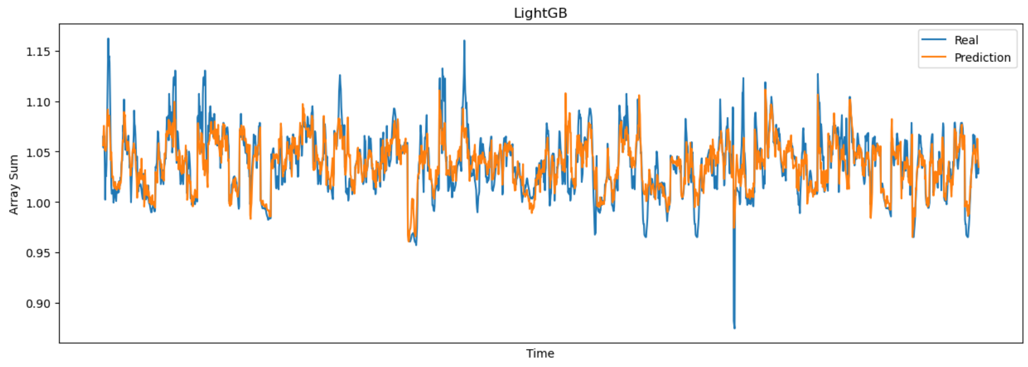
A LSTM neural network with the same architecture than the optimized one and manually selected hyperparameters consisting of two LSTM layers of 48 and 24 units respectively, a dropout layer with a dropout rate of 0.5, and a Dense layer with a number of unit of 24.

	MAPE	MAE	MSE	MRE
animal 1	0.01580	0.01677	0.00057	1.58045
animal 2	0.01253	0.01300	0.00035	1.25399
animal 3	0.01128	0.01142	0.00024	1.12813
animal 4	0.01380	0.01490	0.00045	1.38093
animal 5	0.01003	0.01001	0.00021	1.00389
animal 6	0.01083	0.01119	0.00025	1.08339
animal 7	0.01305	0.01340	0.00035	1.30564
animal 8	0.01018	0.01057	0.00023	1.01879
animal 9	0.01376	0.01409	0.00044	1.37625
animal 10	0.01068	0.01078	0.00025	1.06866
animal 11	0.01156	0.01182	0.00028	1.15681
animal 12	0.01388	0.01433	0.00042	1.38884
animal 13	0.01052	0.01082	0.00025	1.05265
animal 14	0.01201	0.01217	0.00030	1.20119
animal 15	0.01350	0.01381	0.00038	1.35016
animal 16	0.01506	0.01580	0.00050	1.50621
animal 17	0.00880	0.00912	0.00015	0.88090
animal 18	0.00816	0.00872	0.00012	0.81658
animal 19	0.01246	0.01296	0.00034	1.24633
animal 20	0.01292	0.01349	0.00040	1.29249
average	0.01204	0.01246	0.00032	1.20461

Table 5 – Results obtained on test data for all animals with the light gradient boosting machine method, and average results.



(a) All animal 1



(b) Subset animal 1

FIGURE 3.20 – Examples of predictions vs real values with the light GB method. Prediction are made on animal 1.

3.7.6 Result of the comparison

A graphical representation of the results can be found at Fig. 3.21. We can notice that our proposal ("Optimized LSTM" in the figure) obtained the best results in terms of MSE compared to the other methods. This suggests that the models were able to better approximate the true values, and therefore, are more accurate in predicting the outcomes. Additionally, the optimized model also produced accurate results in terms of Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE) compared to the other methods. Only for the MRE did our proposal not obtain the best results. However, even for this measure our proposal outperformed a manually configured LSTM, confirming the importance of the neuroevolutionary phase.

Finally, in figure 3.14 we show the best and worst predictions, obtained on animal 17 and 1, respectively, against the real values. We can see that in both cases the predictions are really close to the actual values, and it is interesting to notice that our proposal could handle the peaks of activity that are more evident in animal 17 then in animal 1.

	MAPE	MAE	MSE	MRE
Gaussian Processes	0.02483	0.02569	0.00135	2.48303
Gradient boosting	0.01324	0.01370	0.00037	1.32489
Random Forest	0.01383	0.01423	0.00041	1.38379
LightGB	0.01204	0.01246	0.00032	1.20461
Non Optimized LSTM	0.01303	0.01794	0.00039	1.94834
Optimized LSTM	0.01224	0.01264	0.00031	1.22479

Table 6 – Average results obtained on test data with Gaussian Process, Gradient Boosting, Random Forest, LightGBM, a LSTM neural network with manually selected hyperparameters and the best model found by the evolutionary process

Overall, the values indicate that the proposed technique has a relatively high degree of accuracy in predicting the output values for the given input data. However, it is important to note that the performance varies across different animals, as indicated by the wide range of values for each metric. This could suggest that the proposed technique may be better suited for some animals than others and that further research may be needed to determine the factors that affect the accuracy of the predictions.

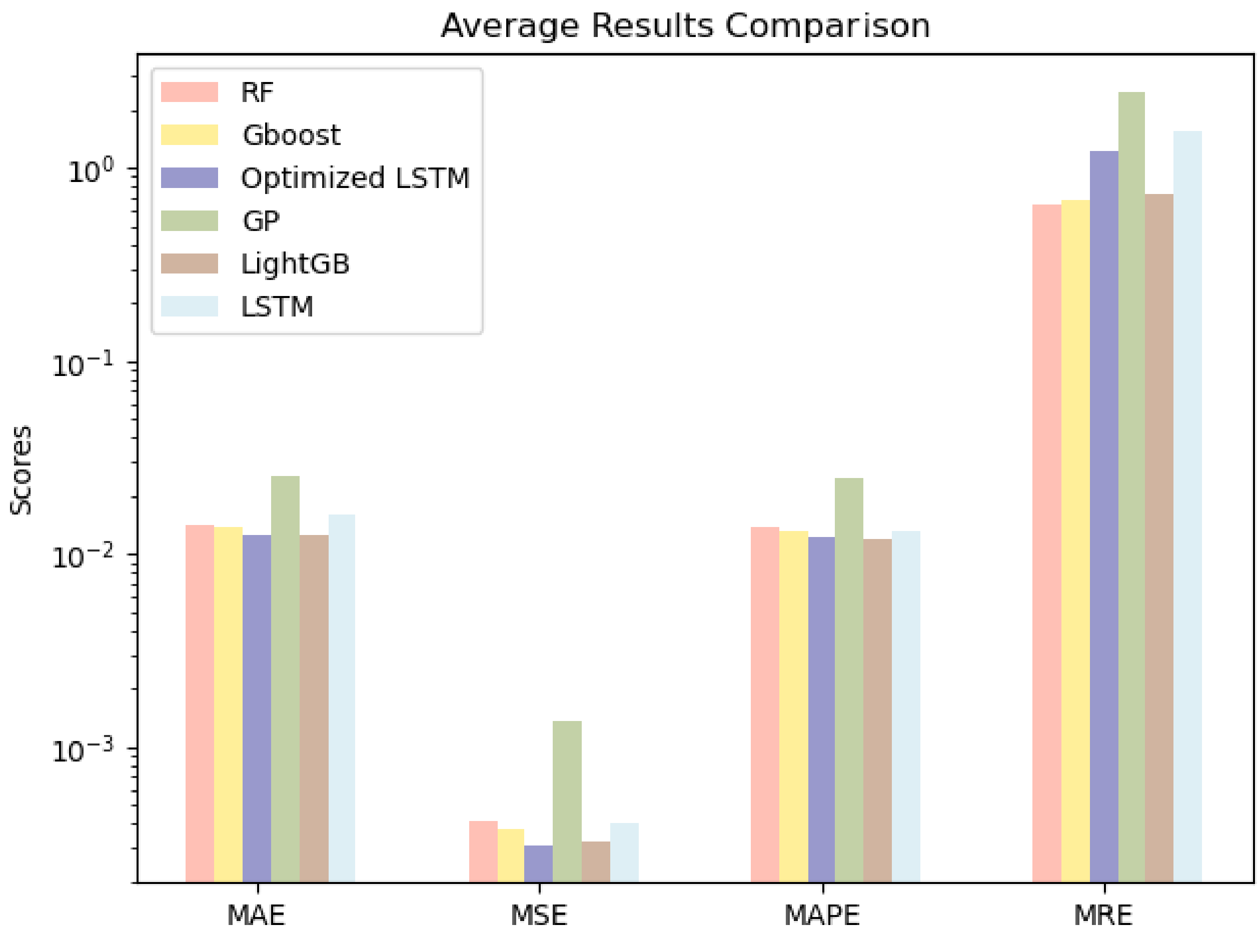


FIGURE 3.21 – Average results obtained on test data with Gaussian Process, Gradient Boosting, Random Forest, LightGBM, a LSTM neural network with manually selected hyperparameters and the best model found by the evolutionary process

4 Discussion

The DENSER methodology has proven to be highly effective in constructing accurate LSTM neural networks and optimizing their hyperparameters, which can lead to improved forecasting results.

By utilizing transfer learning, the pre-trained models can be fine-tuned for specific time series data, which can save time and computational resources while still achieving high accuracy. This transferability also makes the methodology highly applicable in real-world scenarios, where time and resources are often limited.

However, it is important to acknowledge the limitations of the DENSER methodology. The iterative process of training and evaluating multiple neural networks can be time-consuming and computationally expensive, which may limit its practical application in some situations. Additionally, the high time complexity of LSTM models due to their recurrent nature means that training and testing can require significant computational resources, which can be a further limitation.

Future research can focus on enhancing the efficiency of the DENSER methodology to overcome these limitations. For example, using automated machine learning techniques or reinforcement learning algorithms to optimize the model selection and hyperparameters can reduce the computational resources required. Additionally, exploring alternative architectures and techniques, such as hybrid models combining LSTM and other deep learning models, can improve the accuracy of time series prediction and further expand the potential of the DENSER methodology.

Overall, the results presented in this study demonstrate the potential of the DENSER methodology for accurate and efficient time series prediction. Despite its limitations, the methodology provides a promising avenue for future research and practical applications in a variety of fields, including finance, healthcare, and transportation.

Bibliography

[1] Jurgen Schmidhuber Sepp Hochreiter. « LONG SHORT-TERM MEMORY ». In : *Neural Computation* 9(8) (1997), p. 1735-1780.

[2] Eduard Hovy Xuezhe Ma. *End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF*. 2016. URL : https://arxiv.org/abs/1603.01354.

[3] Suresh Manandhar Alexandros Komninos. « Dependency Based Embeddings for Sentence Classification Tasks ». In : *Association for Computational Linguistics* Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (juin 2016), p. 1490-1500. URL : https://aclanthology.org/N16-1175.

[4] Francoise Beaufays Hasim Sak Andrew Senior. « Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling ». In : *Fifteenth Annual Conference of the International Speech Communication Association* (2014), p. 338-342.

[5] Rupesh K Srivastava et al. Klaus Greff. « LSTM: A Search Space Odyssey ». In : *IEEE Trans Neural Netw Learn Syst.* (oct. 2017), p. 2222-2232.

[6] Kenneth O Stanley et Risto Miikkulainen. « Evolving neural networks through augmenting topologies ». In : *Evolutionary computation* 10.2 (2002), p. 99-127.

[7] Reza Pejman et al. « Gradient-based hybrid topology/shape optimization of bioinspired microvascular composites ». In : *International Journal of Heat and Mass Transfer* 144 (2019), p. 118606.

[8] Vicente Coelho Lobo Neto, Leandro Aparecido Passos et João Paulo Papa. « Evolving long short-term memory networks ». In : *Computational Science–ICCS 2020: 20th International Conference, Amsterdam, The Netherlands, June 3–5, 2020, Proceedings, Part II 20*. Springer. 2020, p. 337-350.

[9] David J Montana, Lawrence Davis et al. « Training feedforward neural networks using genetic algorithms. » In : *IJCAI*. T. 89. 1989, p. 762-767.

[10] Nikolaos Gorgolis et al. « Hyperparameter optimization of LSTM network models through genetic algorithm ». In : *2019 10th International Conference on Information, Intelligence, Systems and Applications (IISA)*. IEEE. 2019, p. 1-4.

[11] URL : https://www.tensorflow.org/.

[12] URL : https://keras.io/.

[13] Hussain Alibrahim et Simone A Ludwig. « Hyperparameter optimization: Comparing genetic algorithm against grid search and bayesian optimization ». In : *2021 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. 2021, p. 1551-1559.

[14] Wojciech Zaremba, Ilya Sutskever et Oriol Vinyals. « Recurrent neural network regularization ». In : *arXiv preprint arXiv:1409.2329* (2014).

[15] URL : https://www.tensorflow.org/tutorials/sequences/recurrent.

[16] Michael O'Neill et Conor Ryan. « Grammatical evolution ». In : *IEEE Transactions on Evolutionary Computation* 5.4 (2001), p. 349-358.

[17] Thomas Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

[18] Nuno Lourenço, Francisco B Pereira et Ernesto Costa. « SGE: a structured representation for grammatical evolution ». In : (2016), p. 136-148.

[19] Nuno Lourenço et al. « Structured grammatical evolution: a dynamic approach ». In : *Handbook of Grammatical Evolution* (2018), p. 137-161.

[20] Filipe Assunção et al. « Automatic design of artificial neural networks for gamma-ray detection ». In : *IEEE Access* 7 (2019), p. 110531-110540.

[21] Filipe Assunção et al. « DENSER: deep evolutionary network structured representation ». In : *Genetic Programming and Evolvable Machines* 20 (2019), p. 5-35.

[22] Filipe Assunção et al. « Fast-DENSER: Fast deep evolutionary network structured representation ». In : *SoftwareX* 14 (2021), p. 100694.

[23] Filipe Assunção et al. « Fast-DENSER++: Evolving fully-trained deep artificial neural networks ». In : *arXiv preprint arXiv:1905.02969* (2019).

[24] James Bergstra et Yoshua Bengio. « Random search for hyper-parameter optimization. » In : *Journal of machine learning research* 13.2 (2012).

[25] Jasper Snoek, Hugo Larochelle et Ryan P Adams. « Practical bayesian optimization of machine learning algorithms ». In : *Advances in neural information processing systems* 25 (2012).

[26] Romain Benassi, Julien Bect et Emmanuel Vazquez. « Robust Gaussian process-based global optimization using a fully Bayesian expected improvement criterion ». In : *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5*. Springer. 2011, p. 176-190.

[27] Tinu Theckel Joy et al. « Hyperparameter tuning for big data using Bayesian optimisation ». In : *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE. 2016, p. 2574-2579.

[28] Ziyu Wang et Nando de Freitas. « Theoretical analysis of Bayesian optimisation with unknown Gaussian process hyper-parameters ». In : *arXiv preprint arXiv:1406.7758* (2014).

[29] Jia Wu et al. « Hyperparameter optimization for machine learning models based on Bayesian optimization ». In : *Journal of Electronic Science and Technology* 17.1 (2019), p. 26-40.

[30] H. Hu et al. « Development and application of an evolutionary deep learning framework of LSTM based on improved grasshopper optimization algorithm for short-term load forecasting ». In : *Journal of Building Engineering* 57 (2022), p. 104975.

[31] F. Martínez-Álvarez et al. « Coronavirus Optimization Algorithm: A bioinspired metaheuristic based on the COVID-19 propagation model ». In : *Big Data* 8.4 (22 juill. 2020), p. 308-322.

[32] F. Divina et al. « Hybridizing deep learning and neuroevolution: Application to the Spanish short-term electric energy consumption forecasting ». In : *Applied Sciences* 10.16 (30 juill. 2020), p. 5487.

[33] M. Z. Degu et G. L. Simegn. « Smartphone based detection and classification of poultry diseases from chicken fecal images using deep learning techniques ». In : *Smart Agricultural Technology* 4 (2023), p. 100221.

[34] Y. Wang et al. « Accurate detection of dairy cow mastitis with deep learning technology: a new and comprehensive detection method based on infrared thermal images ». In : *animal* 16.10 (2022), p. 100646.

[35] D. Hadjout et al. « Electricity consumption forecasting based on ensemble deep learning with application to the Algerian market ». In : *Energy* 243 (15 mars 2022), p. 123060.

[36] M. Habibpour et al. « Uncertainty-aware credit card fraud detection using deep learning ». In : *Engineering Applications of Artificial Intelligence* 123 (2023), p. 106248.

[37] Rui Ye et Qun Dai. « Implementing transfer learning across different datasets for time series forecasting ». In : *Pattern Recognition* 109 (2021), p. 107617. issn : 0031-3203.

[38] Stefano Sarti et al. « Under the Hood of Transfer Learning for Deep Neuroevolution ». In : *Applications of Evolutionary Computation: 26th European Conference, EvoApplications 2023*. Springer. 2023, p. 640-655.

[39] AbdElRahman ElSaid et al. « Neuro-evolutionary transfer learning through structural adaptation ». In : *Applications of Evolutionary Computation: 23rd European Conference, EvoApplications 2020, Held as Part of EvoStar 2020, Seville, Spain, April 15–17, 2020, Proceedings 23*. Springer. 2020, p. 610-625.

[40] A. Morteza et al. « Deep learning hyperparameter optimization: Application to electricity and heat demand prediction for buildings ». In : *Energy and Buildings* 289 (2023), p. 113036.

[41] D. Shin et al. « Evolutionary Reinforcement Learning for Automated Hyperparameter Optimization in EEG Classification ». In : *2022 10th International Winter Conference on Brain-Computer Interface (BCI)*. 2022, p. 1-5.

[42] J. F. Torres et al. « Deep Learning for Time Series Forecasting: A Survey ». In : *Big Data* 9.1 (5 fév. 2021), p. 3-21.

[43] Domingo S Rodriguez-Baena et al. « Identifying livestock behavior patterns based on accelerometer dataset ». In : *Journal of Computational Science* 41 (2020), p. 101076.

[44] Adolfo A Rayas-Amor et al. « Triaxial accelerometers for recording grazing and ruminating time in dairy cows: An alternative to visual observations ». In : *Journal of Veterinary Behavior* 20 (2017), p. 102-108.

[45] Mathieu Lepot, Jean-Baptiste Aubin et François HLR Clemens. « Interpolation in time series: An introductive overview of existing methods, their performance criteria and uncertainty assessment ». In : *Water* 9.10 (2017), p. 796.

[46] Holger Teichgraeber et Adam R Brandt. « Time-series aggregation for the optimization of energy systems: Goals, challenges, approaches, and opportunities ». In : *Renewable and Sustainable Energy Reviews* 157 (2022), p. 111984.

[47] Daniel Leite et Igor Škrjanc. « Ensemble of evolving optimal granular experts, OWA aggregation, and time series prediction ». In : *Information sciences* 504 (2019), p. 95-112.

[48] Shanika L Wickramasuriya, George Athanasopoulos et Rob J Hyndman. « Optimal forecast reconciliation for hierarchical and grouped time series through trace minimization ». In : *Journal of the American Statistical Association* 114.526 (2019), p. 804-819.

[49] Yin-Wong Cheung et Kon S Lai. « Lag order and critical values of the augmented Dickey–Fuller test ». In : *Journal of Business & Economic Statistics* 13.3 (1995), p. 277-280.

[50] José F Torres et al. « Deep learning-based approach for time series forecasting with application to electricity load ». In : *Biomedical Applications Based on Natural and Artificial Computing: International Work-Conference on the Interplay Between Natural and Artificial Computation, IWINAC 2017, Corunna, Spain, June 19-23, 2017, Proceedings, Part II*. Springer. 2017, p. 203-212.

[51] Federico Divina et al. « Stacking ensemble learning for short-term electricity consumption forecasting ». In : *Energies* 11.4 (2018), p. 949.

[52] Filipe Assunçao et al. « Towards the evolution of multi-layered neural networks: a dynamic structured grammatical evolution approach ». In : *Proceedings of the genetic and evolutionary computation conference*. 2017, p. 393-400.

[53] Amir Farzad, Hoda Mashayekhi et Hamid Hassanpour. « A comparative performance analysis of different activation functions in LSTM networks for classification ». In : *Neural Computing and Applications* 31 (2019), p. 2507-2521.

[54] Sebastian Ruder. « An overview of gradient descent optimization algorithms ». In : *arXiv preprint arXiv:1609.04747* (2016).

[55] Assunção. *Fast Denser*. https://github.com/fillassuncao/fast-denser. 2019.

[56] Md Abul Bashar, Richi Nayak et Nicolas Suzor. « Regularising LSTM classifier by transfer learning for detecting misogynistic tweets with small training set ». In : *Knowledge and Information Systems* 62 (2020), p. 4029-4054.

[57] Mohamed Marei et Weidong Li. « Cutting tool prognostics enabled by hybrid CNN-LSTM with transfer learning ». In : *The International Journal of Advanced Manufacturing Technology* (2022), p. 1-20.

[58] Shahroz Tariq, Sangyup Lee et Simon S Woo. « CANTransfer: Transfer learning based intrusion detection on a controller area network using convolutional LSTM network ». In : *Proceedings of the 35th annual ACM symposium on applied computing*. 2020, p. 1048-1055.

[59] Christopher Williams et Carl Rasmussen. « Gaussian processes for regression ». In : *Advances in neural information processing systems* 8 (1995).

[60] Jerome H Friedman. « Stochastic gradient boosting ». In : *Computational statistics & data analysis* 38.4 (2002), p. 367-378.

[61] Leo Breiman. « Random forests ». In : *Machine learning* 45 (2001), p. 5-32.

[62] Guolin Ke et al. « Lightgbm: A highly efficient gradient boosting decision tree ». In : *Advances in neural information processing systems* 30 (2017).

[63] Tao Zhang et al. « Improving convection trigger functions in deep convective parameterization schemes using machine learning ». In : *Journal of Advances in Modeling Earth Systems* 13.5 (2021), e2020MS002365.