MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

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

Author: Mónica López-Tafall Criado

Directors: José Portela González,
Jaime Pizarroso Gonzalo

Madrid

Agosto de 2020

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

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

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2019/20 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

[illegible]

Fdo.: Mónica López-Tafall Criado Fecha: 24 / 08 / 2020

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

[illegible]

Fdo.: José Portela González Fecha: 25 / 08 / 2020

[illegible]

Fdo.: Jaime Pizarroso Gonzalo Fecha: 25 / 08 / 2020

COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

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

Author: Mónica López-Tafall Criado

Directors: José Portela González,
Jaime Pizarroso Gonzalo

Madrid

Agosto de 2020

Acknowledgements

Mom, Dad, Pablo, thank you very much for your support, love and understanding throughout these years of University. For being there, even when I wasn't.

To my friends and colleagues because we have walked this path together.

To Santiago Rilo and Daniel Elechiguerra for their contributions worth making this work possible.

To my directors, the chair of Connected Industry and the CIC LAB for the learning opportunities and for bringing me to broaden my horizons.

Mejora de la precisión de las redes neuronales profundas mediante técnicas de preprocesamiento y ajuste de la estructura neuronal. Aplicación a modelos de predicción de series temporales.

Autor: López-Tafall Criado, Mónica

Directores: Portela González, José
Pizarroso Gonzalo, Jaime

Resumen del trabajo

Este proyecto tiene por objeto realizar un amplio examen de las técnicas utilizadas actualmente para aumentar la precisión de los modelos de aprendizaje profundo, prestando especial atención a las que se centran en las redes neuronales, como los algoritmos de selección de variables, las arquitecturas híbridas y las técnicas de optimización de hiperparámetros.

La eficacia de esas técnicas se evalúa posteriormente en una aplicación de caso real

Palabras clave: Redes neuronales, Series temporales, Predicción, Deep Learning

Introducción

La popularización de los modelos de las Redes Neuronales Profundas (*Deep Neural Networks*)en los últimos años ha dado lugar a un aumento del número de técnicas para aumentar la precisión de esos modelos. Sin embargo, es difícil elegir cuál de estos métodos son óptimos para un problema específico. Este proyecto tiene por objeto realizar una comparación de éstos aplicándolos en un problema de previsión de series temporales y proporcionar una metodología inicial que ahorre tiempo y recursos al enfrentarse a un problema nuevo. Otras aplicaciones están fuera del alcance de este proyecto.

Se ha elegido la previsión de series temporales para realizar la comparación debido a la gran demanda de este tipo de modelos Algunas aplicaciones de predicción de series temporales son las tendencias o comportamientos futuros de los mercados bursátiles, las ventas de productos, la demanda de electricidad, la velocidad del viento y la radiación solar para la generación de energía, las cuestiones relacionadas con la salud y el Procesamiento del Lenguaje Natural (PNL), entre muchas otras. Estas aplicaciones han estado, durante muchas décadas (finales del siglo XX), a la vanguardia del desarrollo de algoritmos de aprendizaje automático y de modelos matemáticos complejos en constante evolución [1].

El enfoque de la mayoría de estos modelos de predicción consiste en utilizar los algoritmos de aprendizaje automático para aprender del pasado con el fin de proporcionar una predicción de un valor futuro. Debido a los entornos dinámicos, caóticos, estocásticos y complejos del propio problema y a la incertidumbre de los datos del mundo real (por ejemplo, los mercados bursátiles), esto se convierte en un proceso inherentemente desafiante y no trivial. Esto se agrava cuando se trata de características dependientes del tiempo o de cadenas de información multivariantes a largo plazo, en las que es de suma importancia identificar las correlaciones entre datos temporales distintos [2] o definir qué valores pasados (tanto en tiempo como en variables) son importantes dentro del proceso de predicción [3], respectivamente.

Metodología

El desarrollo de este proyecto ha seguido 5 etapas diferentes:

1. Revisión del estado del arte. Para ello, se revisa en el capítulo 2 la bibliografía disponible sobre los diferentes temas tratados, con el fin de comprender la teoría y los conceptos subyacentes. Esta base teórica ha ayudado a identificar las técnicas más interesantes para aplicar y profundizar en el desarrollo del modelo de previsión de series temporales.

2. Definir el alcance del modelo de predicción. Una vez que se han analizado todas las técnicas potenciales en el estado del arte, se han seleccionado una serie de ellas para evaluar su eficacia en la mejora de la precisión del modelo. El alcance del modelo es lograr la mayor precisión prediciendo la generación de energía eólica del día siguiente en un parque eólico nacional, tomando como entrada datos históricos. Todos los modelos han sido probados con el modelo de referencia, que supone que la energía eólica generada en el día t es igual a la del día t – 1.

3. Investigación de modelos y desarrollo de códigos. A fin de probar todas las técnicas enumeradas, se ha llevado a cabo una investigación exhaustiva de los recursos y bibliotecas disponibles para construir la estructura necesaria para cada uno de los diferentes modelos probados.

4. Validación de modelos y seguimiento de las mejoras. Después de implementar todas las diferentes técnicas de ajuste, se han realizado varias pruebas para ver cómo afecta a la precisión del modelo. También se analizaron características adicionales, como el tiempo de ejecución, para ver si el aumento de la precisión explica la necesidad de una mayor potencia de cálculo, o si con técnicas más sencillas se podría garantizar una solución más equilibrada del problema. La configuración interna de la red también se ha analizado para evaluar la diferencia entre los modelos y utilizarla para comprender mejor la importancia de los hiperparámetros de la red.

5. Resultados y conclusiones. Los capítulos 4 y 5 abordan los resultados y conclusiones derivados de las pruebas realizadas, proporcionando aquellos métodos o técnicas con los que sería más beneficioso trabajar de acuerdo con los datos disponibles, los recursos informáticos y el alcance definido del proyecto.

Resultados

Los resultados obtenidos se pueden dividir por un lado entre las configuraciones que permiten obtener la mejor precisión en la predicción y los que logran resultados en un menor tiempo.

En cuanto al primer punto, el Grid Search, junto a una arquitectura de red tipo GRU es la configuración que mejor precisión obtienen, mientras que los mejores tiempos se obtienen al combinar un Random Search y arquitectura de red tipo RNN.

	Técnica de optimización de hyperparámetros	Capas	Neuronas por capa	Función de activación	Optimizador	Learning rate	Error de validación MSE	Tiempo total de ejecución (min)
LSTM	Grid	2	26	elu	adam	0,00158114	175,390069	260,21
LSTM	Random	3	31	elu	adam	0,00298181	184,265402	45,5
LSTM	Bayesian	3	15	elu	adam	0,00100075	182,650941	118,01
LSTM	Genetic Algorithm	3	30	elu	adam	0,00146076	187,247948	100,37
GRU	Grid	3	16	elu	adam	0,0005	167,600379	217,3
GRU	Random	4	26	elu	adam	0,00096535	169,507542	48,6
GRU	Bayesian	2	30	elu	adam	0,00063344	170,248371	100,14
GRU	Genetic Algorithm	5	39	elu	adam	0,00072304	175,601382	56
RNN	Grid	3	6	elu	adam	0,0005	179,856083	107,1
RNN	Random	1	21	elu	adam	0,00195347	191,678969	24,7
RNN	Bayesian	6	15	elu	adam	0,00073817	185,762487	70,17
RNN	Genetic Algorithm	4	86	elu	adam	0,00156522	176,363621	42
BILSTM	Grid	2	11	elu	adam	0,005	187,233101	511,6
BILSTM	Random	1	41	elu	adam	0,00414321	192,240764	97,3
BILSTM	Bayesian	1	45	elu	adam	0,00259197	190,741036	164,23
BILSTM	Genetic Algorithm	2	69	elu	adam	0,00229161	216,845031	144,07

Tabla 1. Comparación de las mejores simulaciones para cada configuración de red probada.

Si bien las configuraciones de red tipo LSTM se posicionaban como las más interesantes en base a la bibliografía consultada y la cantidad de aplicaciones que se basan en ella, ha quedado demostrado que, para esta aplicación, sus cualidades quedan muy por detrás de las obtenidas con configuraciones de red tipo GRU. Del mismo modo, también se descarta el uso de arquitecturas tipo BiLSTM las cuales no logran mejorar ninguno de los dos principales aspectos contemplados en este trabajo.

En base a los resultados obtenidos se recomienda en cualquier caso optar por una búsqueda inicial de la mejor configuración de los hiperparámetros por medio de un Grid Search, aun cuando el tiempo de ejecución es elevado, y afinar las siguientes búsquedas ya sea usando el mismo método u otro que permita menores tiempos de ejecución. Se recomienda el Grid Search ya que permite una mayor transparencia de los resultados obtenidos, al no estar estos expuestos al riesgo de verse limitados por haber dado con un mínimo local que pueda forzar pruebas de hiperparámetros subóptimas.

Conclusiones

A falta de otros trabajos que confirmen o desmientan las conclusiones extraídas de la aplicación presentada en este proyecto, se asumirá que estas conclusiones son al menos parcialmente aplicables en otro problema de predicción de series temporales. Aun así, se recomienda una búsqueda inicial utilizando la técnica de Grid Search para encontrar el espacio de hiperparámetros óptimo para cada aplicación.

En cuanto a los resultados obtenidos, se pueden resumir las ideas principales en los siguientes puntos:

- Si el tiempo es un factor decisivo en el éxito de la aplicación, se recomienda optar por optimización bayesiana o Random search justo a configuraciones de red tipo RNN. También se pueden combinar junto a arquitecturas tipo GRU si se busca una ligera mejora en la precisión de la predicción del modelo.
- Si el problema se centra en lograr la mejor predicción posible, es decir, el menor error de validación, la mejor opción observada es el uso de un Grid Search junto a una arquitectura de red tipo GRU.
- Si se busca lograr un equilibrio entre tiempo de ejecución y calidad del modelo, optar por optimización Bayesiana en configuración GRU o RNN, siendo preferible la primera.

Referencias

[1] J. Brownlee, 'How to Choose Loss Functions When Training Deep Learning Neural Networks', *Machine Learning Mastery,* Jan. 29, 2019. https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/ (accessed Feb. 08, 2020).

[2] G. Drakos, 'How to select the Right Evaluation Metric for Machine Learning Models: Part 1 Regression Metrics', *Medium*, Feb. 05, 2020. https://medium.com/@george.drakos62/how-to-select-the-right-evaluation-metric-for-machine-learning-models-part-1-regrression-metrics-3606e25beae0 (accessed Feb. 08, 2020).

[3] P. Grover, '5 Regression Loss Functions All Machine Learners Should Know', *Medium*, Feb. 05, 2020. https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0 (accessed Feb. 09, 2020).

Accuracy improvement of deep neural networks through preprocessing and neural structure tuning techniques. An approach to time-series models.

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

Abstract

This project is aimed at conducting an extensive review of current used techniques for accuracy boosting of deep learning models, paying special attention to those focused on Neural Networks, such as feature selection algorithms, hybrid architectures and hyperparameter optimization techniques.
The effectiveness of these techniques is later assessed in a real case application measuring different parameters like the model accuracy and the training time.

Keywords: Neural Networks, Time series, Forecasting, Deep Learning

Introduction

The popularization of Deep Neural Network models in recent years has led to an increase in the number of techniques to boost the accuracy of these models. However, it's difficult to choose which of these method are optimal for a specific problem. This project aims to perform a comparison of these by applying them in a time-series forecasting problem and provide an initial methodology that saves time and resources when faced with a new problem. Other applications are out of the scope of this project.

Time-series forecasting has been chosen to perform the comparison due to the high demand of this type of models. Some applications of time-series forecasting are future trends or behaviors of stock markets, product sales, electricity demand, wind speed and sun-irradiation for power generation, health- related issues and Natural Language Processing (NLP), among many more. These applications have, for many decades (late 20th Century), been at the forefront of development of machine learning algorithms and ever evolving complex mathematical models [1].

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

[2] or to define which past values (both in time and variables) will be considered within the prediction process [3], respectively.

Methodology

The development of this project has followed 5 different stages:

1. State of the art review. To this end, the available bibliography on the different covered topics is reviewed in CHAPTER 2. in order to gain understanding of the underlaying theory and concepts. This theoretical basis has helped in identifying the most interesting techniques to apply and deep dive into while developing the time series forecasting model.
2. Define the scope of the forecasting model. Once all the potential techniques have been analyzed in the state of the art, a series of them have been be selected in order to assess their effectiveness in improving accuracy metrics. The scope of the model is to achieve greatest forecasting accuracy for next-day wind power generation in a national wind farm, taking historical data as input. All models have been tested against the benchmark model, which assumes Wind Power at day t is equal to day t-1.
3. Model research and code development. In order to test all the listed techniques, an exhaustive research on available resources and libraries to build the required structure for each different tested model has been conducted primarily.
4. Model validation and improvement tracking. After implementing all the different tuning techniques, several tests have been performed to see how accuracy is affected. Also, additional features were analyzed, such as performance time, to see whether the increase in accuracy accounts for a need of greater computing power, or if simpler techniques could warrant a better-balanced solution to the problem. Internal net configuration have also analyzed to assess difference between models and use it to further understand network hyperparameters importance.
5. Results and conclusions. CHAPTER 4. and CHAPTER 5. address the results and conclusions driven from the conducted tests, providing those methods or techniques that would be most beneficial to work with according to the available data, computing resources, and defined scope of the project.

Results

The results obtained can be divided between the configurations that allow the best accuracy in prediction and those that achieve results in a shorter time.

Regarding the first point, the Grid Search, together with a GRU type network architecture is the configuration that obtains the best accuracy, while the best times are obtained by combining a Random Search and RNN type network architecture.

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

Table 1. *Cross comparison for the best performing simulations*

Although LSTM type network configurations were positioned as the most interesting ones based on the consulted bibliography and the amount of applications based on it, it has been demonstrated that, for this application, its qualities are far behind those obtained with GRU type network configurations. Likewise, the use of BiLSTM-type architectures is also discarded, as they do not improve any of the two main aspects contemplated in this work.

Based on the results obtained it is recommended in any case to opt for an initial search of the best configuration of the hyperparameters by means of a Grid Search, even when the execution time is high, and to refine the following searches either using the same method or another one that allows shorter execution times. An initial Grid Search is recommended because it allows a greater transparency of the results obtained, since they are not exposed to the risk of being limited by having found a minimum local that can force suboptimal hyperparameter tests.

Conclusions

As a synthesis of this work, it can be concluded that the tests carried out in this work cannot be considered conclusive in the absence of other studies that can praise or complement what is observed here, but, although it is understood that not all prediction problems are to behave as observed in this work, the results are considered to be partially reproducible and applicable to problems with similar characteristics.

As for the results obtained, the main ideas can be summarized in the following points:

- If time is a decisive factor in the success of the application, it is recommended to opt for Bayesian optimization or Random search just to RNN type net configurations. They can also be combined with GRU-type architectures if a slight improvement in the model prediction accuracy is sought.

- If the problem is focused on achieving the best possible prediction, that is, the lowest validation error, the best option observed is the use of a Grid Search together with a GRU-type network architecture.
- If a balance between runtime and model quality is sought, opt for Bayesian optimization in GRU or RNN configuration, the former being preferable.

References

[1] J. Brownlee, ‘How to Choose Loss Functions When Training Deep Learning Neural Networks’, *Machine Learning Mastery*, Jan. 29, 2019. https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/ (accessed Feb. 08, 2020).

[2] G. Drakos, ‘How to select the Right Evaluation Metric for Machine Learning Models: Part 1 Regression Metrics’, *Medium*, Feb. 05, 2020. https://medium.com/@george.drakos62/how-to-select-the-right-evaluation-metric-for-machine-learning-models-part-1-regrression-metrics-3606e25beae0 (accessed Feb. 08, 2020).

[3] P. Grover, ‘5 Regression Loss Functions All Machine Learners Should Know’, *Medium*, Feb. 05, 2020. https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0 (accessed Feb. 09, 2020).

TABLE OF CONTENTS

TABLE INDEX

TABLE OF FIGURES

CHAPTER 1. INTRODUCTION

1.1 Motivation

The realization and context of this project is based on an aim to carry an in-depth review of current trends regarding time series forecasting, exploring different architectures of Deep Neural Networks (DNNs) based models.

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

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

1.2 Objectives and Scope

The overall objective of this work is to deep dive into the principles that rule some of the most widely spread Deep Learning application techniques and provide useful guidelines that could be used when facing a novel prediction problem.

When faced with a new problem of time series prediction, one of the biggest challenges is to choose the most appropriate model to, at least, start iterating and improving the model to make it fit as well as possible to the available data.

This choice is not trivial, and its complexity depends in part on the resources available, including time. It may be that the application requires obtaining the best possible prediction model, at the expense of execution time. It may be that time is a key factor in the application and that the models, therefore, must be able to be executed under strict time requirements. Or you may simply want to start an initial search process for the most suitable hyperparameters with a fast and simple model that allows you to fine tune them by means of a more complex model afterwards, without getting lost in the complex initial search.

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

CHAPTER 2. ABOUT ARTIFICIAL NEURAL NETWORKS. STATE OF THE ART

1.1 Brief introduction to Artificial Neural Networks

Artificial Neural Networks, also known as Neural Networks (NN), emerged as a means of mimicking the biological, complex behavior of the human brain.

Artificial neurons are modeled to work just as biological neurons, where a series of inputs (with respective weights associated to each one, called synaptic weights) are "summarized" (Dendrites) and passed on to an activation function that processes the information (Nucleus), after which an output is generated (Axon).

A single neuron (Perceptron) does not perform well enough, but further development lead to the wiring of multiple artificial neurons (Synapses), so that the output of one became the input of the next one. This technique creates multiple layers that would manage to provide much more accurate results and acquire a much higher resemblance to the human brain behavior. This is called an Artificial Neural Network.

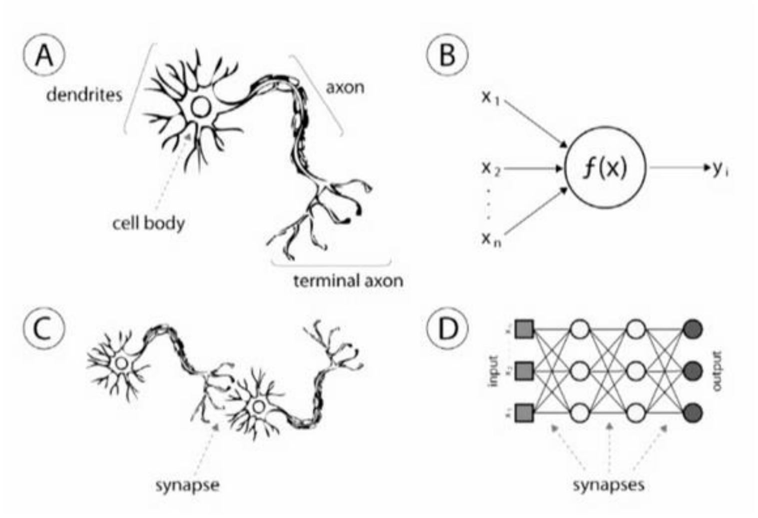


Figure 1. Correlation between biological neurons and ANN. Source:[4]

Formally speaking, the three processes described ahead are commonly known as Input layer - the one in charge of receiving and passing on the parameters for the model - , the Hidden Layer(s) – A feed forward network where a wide variety of machine learning algorithms can be employed to perform multiple operations- and the Output layer – where a single output is given as the prediction - [5]. But, of course, over the course of time this process has been renewed, optimized and further developed to try to boost the prediction accuracy, leading to a vast portfolio of prediction models.

This portfolio includes examples such as the Multi-Layer Perceptron (MLP), Deep Learning NNs, Recurrent NNs, Long-Short Term Memory NNs (LSTM), Convolutional NNs, Recursive

NNs, etc. whose application depends mainly on the data provided and expected output, although variations of their usage can be found in the available literature.

A general classification of these models can be found below [6], although this might differ from real applications in which, sometimes, multiple architectures are combined with the objective of targeting various problems within the same scope (classification problems based on Time series forecasting) or boosting accuracy..

- Unsupervised Learning: Extracts patterns from a set of unlabeled data
  - Restricted Boltzmann Machine
  - Autoencoder
- Classification
  - Text Processing tasks like sentiment analysis, parsing (grammar analysis like identifying substantives, verbs, subjects, …) or named entity recognition.
    - Recurrent Neural Network (Character Level)
    - Recursive Neural Tensor Network
  - Image recognition
    - Convolutional Neural Network
    - Deep Belief Network
  - Object recognition
    - Convolutional Neural Network
    - Recursive Neural Tensor Network
  - Speech recognition
    - Recurrent Neural Network
  - Other general tasks
    - Deep Belief Network
    - Multi-Layer perceptron with ReLU activation function
- Forecasting
  - Recurrent Neural Network

For each different problem (stocks prediction, Natural Language Processing, Image and Video recognition/classification, wind speed forecasting, …), the aim is to provide a specific model with meaningful data in order to obtain a meaningful prediction. This is achieved by means of training a model, i.e., teach a model what is the proper output for each input data. The model should extrapolate the correct output when facing similar data that has not been used in this training process. Therefore, we want to create rules associated to each problem that the models can use to generate an output accordingly to the provided information, for each given task.

In the end, the objective is to adjust properly the weights associated to each input variable so that the loss function is optimized, that is, the model is able to achieve the minimum value for its loss score or distance between the predictions ($\hat{y}$) and the true target values (y), during the training phase. This is done by the *optimizer*, which typically implements the *backpropagation* algorithm that enables to calculate the gradient descend or, in other words, how much does the loss function change given a change in the weights. The discovery of this algorithm is what originally boosted the popularity of Neural Networks.

2.1.1 Loss Functions

Some of the most widely used loss functions are detailed as follows:

MSE

Mean Square Error (MSE) is the most used loss function for regression problems. It is calculated as the average of the squared difference between the predicted values (O_i) and the actual or expected values (E_i). The squaring allows for large differences to make a greater impact of the loss score than the smaller ones [7]. This allows the optimizer to focus on the values that acquaint for these larger differences and try to modify the corresponding weights in order to correct (minimize) the loss score in the following iterations.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(E_i - O_i)^2$$

One issue related to this loss function is that a single very bad prediction would cause the loss score to escalate, not giving a realistic image on how well the model is performing, had it no been for that specific bad prediction. Same can have when a lot of small errors occur. When this happens, the loss score will be low due to the small distances, which will cause to underestimate the model's bad performance [8].

RMSE

The Root Mean Square Error is, as it name estates, the root of MSE. By computing the square root, the scale of the errors can be compared withing the same scale as the target variables, which makes it easier to understand at first sight.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(E_i - O_i)^2}$$

Although both RMSE and MSE are very similar, there are some differences in the way it affects the model performances (if based on gradient descend) that need to be understood or take into consideration when using one or the other. Both methods are equally usable with regards to models scoring, but not directly interchangeable for gradient-based models [8]. This is due to the fact that both functions resemble each other in terms of gradient descend, but with different flowing rates and the flowing rate depends on the loss score of each function. As the square root affects the loss-score result, the flow rate is also modified and so in order to interchange both RMSE and MSE in the same model, it would be necessary to modify accordingly some hyperparameters like the learning rate[8].

MAE

The Mean Absolute Error (MAE) function is adequate on some cases in which the distribution of the endpoint resembles a Gaussian distribution but presents some larger or smaller values far for the mean value (outliers).

The MAE loss score can be obtained by computing the average of the absolute difference between Predicted and Expected values.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|O_i - E_i|$$

Unlike MSE, this loss function is not as sensitive to outliers and, therefore, doesn't penalize big prediction errors so badly, allowing to have a more realistic value on how well the model performs in the presence of outliers. However, using this loss function can also limit the model's performance due to its large gradient. The fact that the gradient is not modified, even when approaching a minimum value (nearly 0 loss score), makes is difficult to find the solution, which might be skipped unless a dynamic learning rate is used, adding complexity to the model.

MSE, might not perform well in presence of outliers, but its gradient decreases when approaching the minima, even with a fixed learning rate, making it more precise when training the model.

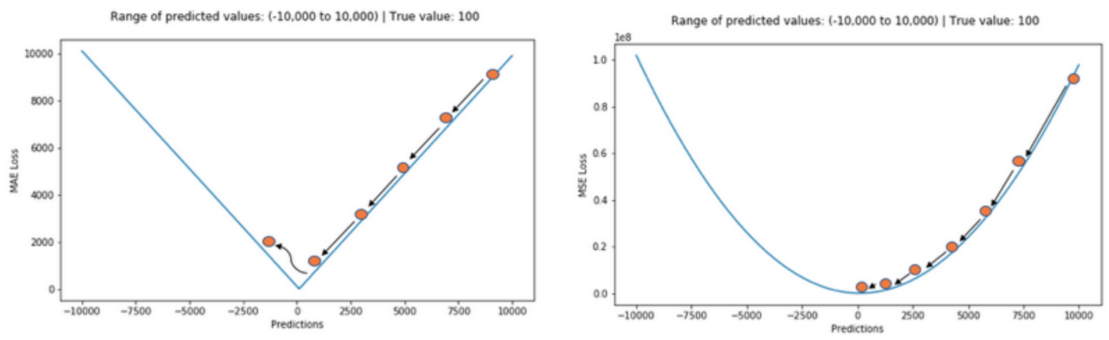


Figure 2. MAE and MSE gradient descend, Source:[9]

$\underline{R^2}$

The R^2 function is actually a ratio that gives a metric on how good the model is compared to the naïve mean model. This is, the model MSE is compared to the baseline model MSE, being the baseline model that in which the prediction will always be the mean of all samples[8]. The R^2 values range from -∞ to 1, where values closer to 0 indicates a model very close to the baseline (the model is unable to outperform the baseline) and 1 indicates a model with almost no error.

$$R^2 = \frac{MSE(model)}{MSE(Baseline)} \qquad MSE\ (baseline) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y}_i)^2,$$

Being y_i the real value and y_i the mean of the observed value.

Huber Loss

The Huber loss function, also known as Smooth Mean Absolute Error, is similar to MAE but squares it value when the error is small. In order to determine how small must that error be, a δ hyperparameter is tuned. When δ is close to 0, the Huber Loss score is very similar to MAE and will resemble MSE score when δ takes a large value (∞).[9]

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}\left(y, f(x)\right)^2, & for\ |y - f(x)| \le \delta \\ \delta|y - f(x)| - \frac{1}{2}\delta^2, & otherwise \end{cases}$$

The value given to δ determines what it considered to be an outlier in the problem, therefore limiting the sensitiveness of the problem in presence of outliers. When residuals are smaller that δ, they will be treated as with MSE, whereas larger values will be minimized according to MAE which is less sensitive to these residual values.

The main advantage of using this loss function is that it combines the advantages of both MAE and MSE. It is more robust to outliers than MSE and curves around the minima, decreasing the gradient. The one problem of using Huber Loss is that the δ value needs to be adjusted just as any other hyperparameter of the model (see section 3.3.3), which takes time and adds some difficulty to the training process.

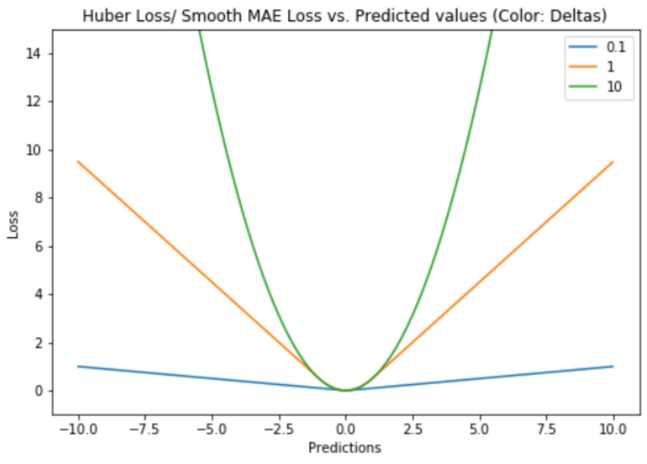
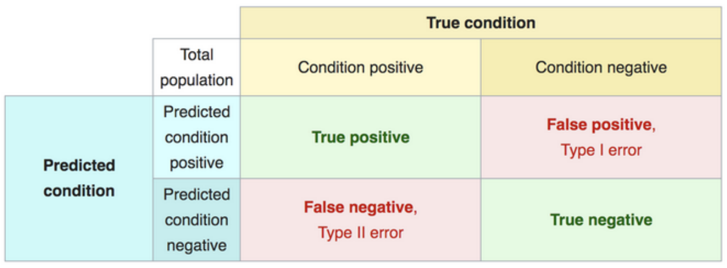


Figure 3. Huber loss plot. Source:[9]

F1-SCORE

The F1-Score is mostly used to asses accuracy in classification problems. The F1-Score bases its inner operation in the results obtained from the confusion matrix, generated after classifying a series of labels.



		True condition	
	Total population	Condition positive	Condition negative
Predicted condition	Predicted condition positive	**True positive**	**False positive,** Type I error
	Predicted condition negative	**False negative,** Type II error	**True negative**

Figure 4. Confusion matrix. Source:[10]

Based on these values, it is computed the precision of the model (percentage of actually real positive values (TP) among those predicted to be positive (TP+FP)) and the recall (percentage of predicted positive (TP) values among those that really are positive (TP+FN)). The harmonic mean of these two values is the F1-Score value[10].

$$F_1 \frac{2}{(Recall)^{-1}(Precision)^{-1}} = \frac{2}{\left(\frac{TP}{TP+FN}\right)^{-1}\left(\frac{TP}{TP+FP}\right)^{-1}}$$

Information about alternative loss functions, for both classification and regression problems, can be found in [9], [11], [12].

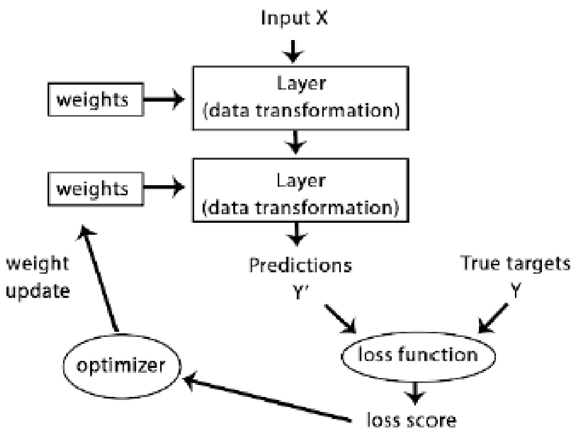


Figure 5. Weights adjustment using optimizer. Source: [13]

Once the network has been properly trained, it is expected that the model can behave with such diligence when dealing with new data (test/validation phase).

1.2 Application examples

In this section, a review of different applications using NNs is provided based on various research papers.

For all of them, information regarding application sector, input and output characteristics, and information regarding architecture and any other relevant features of each prediction model have been extracted in order to identify the most generally used methods and those tools that have proved useful when applied to similar problems to those we are addressing in this work.

As a brief summary of all the information contained in the tables below, and having conducted an intense research in order to identify the most interesting pieces of information, it seems fair to claim that LSTM are the most widely used methods when developing prediction models in which past observations and trends play an important role in future behaviors.

Other algorithms, such as those based on Convolutional Neural Networks, also have their place when dealing with this issue, although their use and deployment has been mostly limited to Natural Language Processing and Image classification. As they were developed for classification purposes, their application in time series forecasting has mostly been explored in decision models, where a binary output is expected, depending on whether it would be most interesting to buy or sell.

Moreover, a model combining CNN and bi-directional LSTM claims to achieve a 9% increase in prediction accuracy, when compared with single deep learning models, and an over 6 times increased accuracy regarding SPM models.

Additional algorithm features such as Feature selection, Genetic Algorithm and exponential smoothing or normalization have been explored in order to reduce model complexity, optimize both architecture and input selection aspects and reduce random variations and noise, naturally present in time series data.

Project	Sector	Method	Input	Output	Accuracy	Metric	Batch size	Architecture information	Additional information
Sector stock Price analysis [2]	Finance	DWNN (Deep and Wide NN)	12 stocks with 5 different features 60-day historical data as input Jan 2000 to Aug 2017	7 days prediction window for all 12 stocks	0,057443 30% error reduction compared to general RNN model (0,0809361)	MSE	Random sampling method to extract sample with the size of the batch as input	12 seq2seq models GRU as RNN cell CNN layer every 5 time steps (1 CNN layer= 2x Convolution Layers + Pooling layer+ 1x Full connection Layer)	Model implemented using TensorFlow Train: 01/01/2000 to 30/12/2015 Test: 01/01/20016 to 16/08/2017
Stock prediction [14]	Finance	Paragraph vector + LSTM	Text (News) + 50 company's closing stock prices	Numerical 10 company's closing stock prices	-	-	30 minibatch size	50 epochs 1 layer – 20 steps 50% dropout in non-recurrent connections	No information given about past observation period considered
Neural Machine translation [15]	NLP	LSTM + attention module	Text	Text	-	-	-	DEEP LSTM 8 encoder and 8 decoder layers Attention connections from decoder to encoder	-
Stock Markets Price movement prediction [16]	Finance	LSTM	Numerical 180 features	Binary 01 – sell/buy 15 minutes prediction window - 1 stock prediction at a time	55,9% Kruskal Wallis to compare accuracy improvements between models	$\frac{tp+tn}{tp+fp+tn+fn}$ Tp= true positive Tn= true negative Fp= false positive Fn= false negative	-	-	Exponential smoothing applied through exponentially weighted moving average – reduce random variation and noise Training - 10 months Validation – 1 week Output activation function: *tanh*
Stock returns predictions [17]	Finance	LTSM	Multivariate Numerical values 30-days long sequences with 10 features and 3 days earning rate labelling	Numerical	14,3%-27,2% Normalizatio n very useful for improving accuracy	-	-	See comment (i)	2013-2015 → 1211361 sequences Training: 900000 seq. Validation: 311361 seq.
Electric load forecasting [18]	Electric	LSTM + GA + FS	Multivariate numerical data (Demand, weather and time lags) Jan 2008-December 2016	Various time horizons considered	353.38	RMSE	150	Activation = relu, weight optimization = adam, number of epochs = 300, learning rate = 0.005	GA to find optimal time lags and number of layers Data normalization in range [0,1] via feature scaling 70/30 train-val ratio Need for stationary TS

Project	Sector	Method	Input	Output	Accuracy	Metric	Batch size	Architecture information	Additional information
Stock prediction [19]	Finance	CNN	Multivariate Numerical values Five stock features from jan 2015 to dec 2017	Binary 01 – sell/buy 10 days prediction window	60%	F1-score	Batches of 1000 samples	Conv1D adapts our time series information to conv networks internal function Relu-Lrelu activation functions Max pooling function required to reduce feature dimension Softmax function for output layer See comment (ii)	Noise reduction is needed Min-max normalization 80/20 train-val ratio
BI-Directional LSTM for stock market prediction [20]	Finance	Bi-directional LSTM + CNN	Univariate numerical values 10 year S&P 500 index data segmented into sequences of 50 closing data prices	Numeric value of predicted closing price for seven days into the future	0.000281317	MSE	50	See comment (iii) Adadelta optimizer	Relative change normalization
Short term wind power prediction [21]	WPP	Multi Layer Restricted Boltzmann Machine (MRBM) – Deep Belief Network	1200 hours of data from a wind farm with Wind power and Wind Speed features data	Wind power values for the next 4 hours with a 15 minutes time resolution	0,172 0,123 0,698	RMSE MAE RC (Relative Coefficient)	-	3x RBM layers With a [350,200, 300] nodes structure	Train: 10542 samples Test: 2639 samples Better error distribution than compared to a BPNN model
Stock price movements and trading strategies [22]	Finance	Attention based LSTM model	Stock Prices Data from Taiwan Stock Exchange (5 features) and other Technical indicators are then calculated(KD,MA,RSV,…)	Multiclass output Class 0 -stock price increase of more than 3%, Class 1 - increase of 2-3% Class 2 - increase of 1-2% Class 3- increase of 0-1% Class 4 -flat stock price Class 5- stock price decrease of 0-1% Class 6 - decrease of 1- 2% Class 7- decrease of 2- 3% Class 8 - decrease of more than 3%	-	Accuracy= $\frac{tp+tn}{tp+fp+tn+fn}$ Precision = $\frac{tp}{tp+fp}$ Recall= $\frac{tp}{tp+fn}$ F1 Score = $2\frac{P*R}{P+R}$	-	Input layer (dimensionality features: technical indicators and prices data)+ *tanh* act.function Attention layer Output layer + *Softmax* act..function	Attention mechanism to avoid vanishing gradients when dealing with long-term dependencies Model deployed using TensorFlow

Project	Sector	Method	Input	Output	Accuracy	Metric	Batch size	Architecture information	Additional information
Green House Gas analysis [23]	Health & Environment	Adaptative NeuroFuzzy Interference Systems (ANFIS) RNN LSTM	16x 2921 time series of GHG concentration (Data points in each TS are spaced every 6 hours) 10th May - 31st July 2010	-	ANFIS 0.103294 0.045501 RNN 0.101140 0.048277 LSTM 0.100480 0.043279	RMSE MAE	-	ANFIS model 4 regressors, 12 rules, 0,002 learning rate, 200 epochs, Adam learning algorithm, loss function: MSE RNN & LSTM Models 4x Input layers, Adam optimization algorithm, 20 iterations max., loss function: MSE (The difference between RNN and LSTM is that the corresponding nodes are used in the hidden layer (RNN OR LSTM cell).)	Differencing to make data stationary and normalization to achieve values in a [0,1] interval Three different models deployed using TensorFlow and Keras
C-Reactive protein concentration prediction [24]	Health & Environment	LSTM	10 patient's data recorded over past 250 days	Next time step prediction for each patient	5 HL- 2,0678 50 HL- 1,9952 100 HL- 1,6876 200 HL- 1,4303 300 HL- 1,8667	RMSE	-	Same architecture with variable number of hidden layers (5,50, 100, 200 and 300)	90/10 train- val ratio Standardized data to have zero mean and unit variance
Early smoke detection for forest wildfire video [25]	Health & Environment	Restricted Boltzmann Machine + MLP (Deep Belief Network)	Smoke video divided into 482 frames (resized to 16 16 3)	Smoke Yes/no	95%	Detection Rate= $\frac{tp - fp}{total\ smoked\ frames}$	Mini-batches	RBM used for dimension reduction (2 layers – Visible +Hidden layers) + MLP as output layer (Stochastic Gradient Descent method) Learning rate= 0,0006 512 nodes 2 hidden layers 100 epochs 100 iterations max	17/30 train-val ratio Data is divided into 100 subsets
Eletric load forecasting in Smart Grids [26]	Electric	LSTM	Electric load data set electricity consumption every 15 minutes (10 past days → 904 data samples) International Airline passengers- monthly totals → 144 observations (12 years)	Electric load forecast- 96 time steps ahead prediction (next day) International Airline passengers – 12 months time window	Electric load forecast 0,0702 0,0535 International Airline passengers 0.0435 0.0345	RMSE MAPE(Mean Absolute Percentage Error)	-	-	-

Project	Sector	Method	Input	Output	Accuracy	Metric	Batch size	Architecture information	Additional information
Multivariat Time Series Prediction Framework [27]	-	Low-High Convolutional Neural Network (LHCnn) CNN+Multi step Attention mechanisms	Traffic data set Occupancy rates measured hourly by 862 sensors Recorded in 2015-2016 (36 hours input window size) Solar-Energy dataset Data recorded from 137 PV plants every 10 minutes during 2006 (12 input window size)	Traffic data set 4 different prediction horizons (3,6,12 and 24 hours) Solar-Energy dataset 30 to 120 minutes prediction window	Traffic data set – 3 hours 0,4547 0,3209 0,8823 Solar-Energy dataset – 30 minutes 0,0944 0,0492 0,9960	RMSE RAE Empirical Correlation Coefficient(CORR)	-	-	Residual connections from the input to the output of the block to avoid gradient explosion/diffusion Multi-step Attention used to build connections between Low-level and High level convolution structures. Linear mappings used to ensure that the output size matches the input in convolution 60/20/20 – Train- Val- Test ratio
Stock Price forecast [28]	Finance	Back propapagtion Feed Forward NN + Discrete Wavelet Transformation (DWT)	Apple stock prices recorded during May 2008 until may 2018 – 2520 datapoints in total 2 approaches to input data are considered. 1. Input data set contains 8 business days 2. Input data contains 4 business days + 4 weekly average values of one-month resolution	5 business days prediction with weekly shift	1ST data set Model 1- 3,55 0,96 Model 2- 3,29 0,95 2nd data set Model 1- 3,60 0,97 Model 2- 4,35 0,95	RMSE R	-	Model 1. BPNN with 1 hidden layer 5x16x5 (number of neurons in input, hidden and output layers) + ReLU Model 2. BPNN with 2 hidden layers 5x16x8x5 (number of neurons in input,1st hidden, 2nd hidden and output layers) +ReLU	70/30 train-Val ratio DWT is used to decompose the time-series data into discrete wavelets, eliminating the noise effect. The Haar function is used as the wavelet basis function.

1.3 Accuracy boosting techniques

Having spoken about *meaningful* inputs, *activation functions*, *optimizers*, different NNs models, application examples and so on in Section CHAPTER 2. , and looking at the information provided in Section 1.2, it is easy to perceive the complexity behind the deployment of these prediction or classification models, even more when managing the necessity of having to fine-tune them in order to boost accuracy for a given problem.

If it is already difficult to choose a model when a new problem appears, different research papers have proven that combining different neural architectures produces a considerable increase in the accuracy of the model. This trend has led to the outcome of hybrid models such as The Deep Belief Network (Restricted Boltzmann Machine + RNN/LSTM), Seq2seq models (a combination of two separate Recurrent Neural Networks), Deep and Wide NNs (CNN+ RNN), Bi-directional NNs, Low-High CNNs and many more that researchers keep developing over the years.

In addition to choosing the right architecture, as a combination or not of different models, comes the need to adjust the number of hidden layers, number of units, activation functions, optimizer, epochs, dropout, batch size, learning rate, number of iterations, and many more hyperparameters whose tuning is not trivial nor immediate or dependent on the expected output [5]. How do these affect the model's performance? To which extent are we manipulating the output?

And what about the already mentioned *meaningful* inputs? How can we be sure that we are giving the model the correct information to deal with?

Variable selection is one of the most critical steps for achieving a high degree of accuracy. When dealing with large data sets it is complicated to choose effectively those features that are not only uncorrelated among them, but also those that can manage to retain valuable temporal information. In recent years, resources such as Genetic Algorithms, Harmony Search, Temporal Memory search, attention modules, Paragraph Vector, etc. have emerged to:

1. Improve feature selection to assess the use of relevant not correlated input data that bring value to model operation
2. Helps with vanishing gradient problem when in need to handle long-terms dependencies.

In order to explain how any of the mentioned variations over traditional NNs models can improve accuracy, we will rely on available bibliography and explain its results based on already trained examples.

1.3.1 Variable selection models

Variable or feature selection is based on different algorithms that seek to automatically select attributes from the available data that are most useful for the predictive problem we might be dealing with.

This method is similar to dimensionality reduction as it enables to lower the number of inputs passed along to the model, although the means of doing so vary between these two methods.

Whereas dimensionality reduction algorithms achieve attribute reduction by generating new combinations of the initial attributes (Principal Component Analysis, for example), feature selection algorithms manage to do so by eliminating attributes that are not useful, redundant or unwanted. Therefore, the use of Feature Selection (FS) has three main advantages [18], [29], [30]:

1. Improves prediction accuracy by guaranteeing that only useful information will be processed in the network. This also helps to reduce overfitting as there will be less chance of learning from noise present in the data set.
2. Reduces computation weigh by leveraging the amount of information given to the predictive model
3. Provides better understanding of the underlying process and enables to achieve more simple and explainable models. There is an inverse correlation between the amount of data attributes and the explainability of a model.

Feature selection methods can be classified according to these three categories:

A. Filter based FS algorithms

Filter based FS algorithms use statistical metrics or techniques to filter features. This filtering is based on the relationship between each input variable and the target variable, according to the chosen metric.

Filter based techniques evaluate each predictor regardless of the forecasting model, giving each data column a feature score and ranks them by their predictive importance. The predictive model will later on use only those (n) variables that pass the established criteria whereas the left over variables are completely discarded. [31].

The main problem related to filter-based methods is that they are mostly univariate techniques, which means that each predictor is only evaluated regarding the output variable, discarding any possible interactions with other input variables. This, in turn, may lead to the training of the model with redundant, yet relevant, information, causing collinearity problems to appear. Also, some of these techniques only consider linear dependence between variables, increasing the risk of eliminating important features as their real correlation with the output variable might not be identified [3], [31].

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

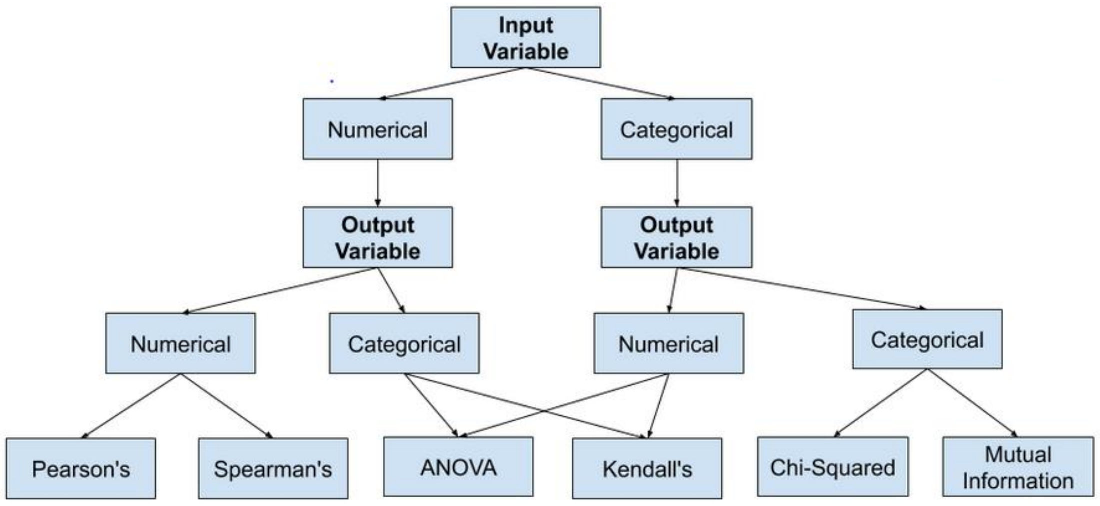


Figure 6. Filter based FS methods. Source: [31]

A1. Pearson's correlation coefficient (Linear)

Also known as the R value in statistical models, the Pearson's Coefficient returns a value between ± 1 (where -1 indicates a strong negative correlation and + 1 a strong positive correlation) that grades the level of correlation, or linear degree of relationship, between two variables (x and y). This value is computed by dividing the covariance of both variables by the product of their standard deviation [32].

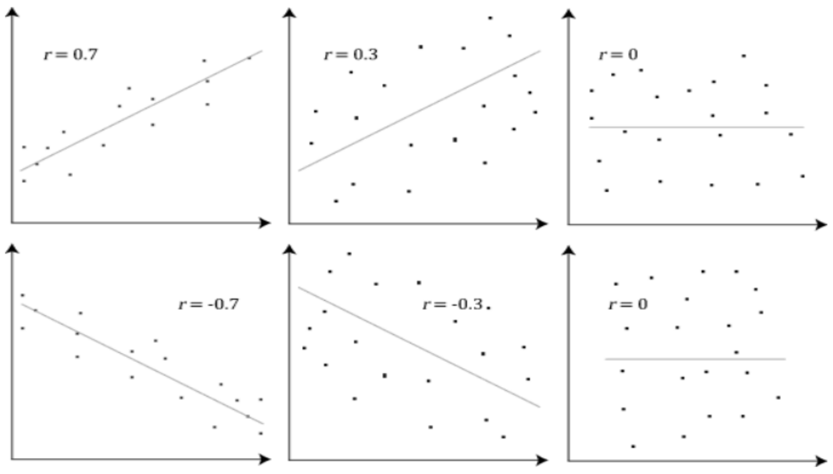


Figure 7. Pearson correlation values. Source: [33]

$$r = \frac{\sum(x-\bar{x})(y-\bar{y})}{\sqrt{\sum(x-\bar{x})^2}\sqrt{\sum(y-\bar{y})^2}}$$

This method requires both variables to be measured in an interval or ratio scale, although it is not required that both variables need to be comprised in the same scale or units, and it is not affected by changes in the scale of either variables. Other assumptions and requirements for computing the Person's Coefficient are [33]:

a. Variables must approximate to a normal distribution (data points waver over the mean value)

b. Data distribution shows homoscedasticity. That is, by looking at the scatter plot, the points must lie equally on both sides of the Fitness line. If the scatter plot shows a cone-like distribution, homoscedasticity would not be guaranteed.
c. The two variables must have a linear relationship. If it is known beforehand that this requirement is not fulfilled, the Pearson's Coefficient will not be able to capture the dependency of both variables.
d. The data has been treated for outliers as these can make the correlation coefficient inaccurate.
e. Both variables need to have the same amount of observations.
f. Both variables must be continuous.

A2. Spearman's coefficient (Non-linear)

This is the nonparametric version of the Pearson's coefficient, also known as the rho (ρ) coefficient. This method allows to measure the degree of relationship between two variables, as well as their monotonic relationship, and is indicated when dealing with ordinal data or in cases in which any of the conditions for using the Pearson's coefficient are violated. Once calculated, it returns a value between ± 1 , as in with the Pearson's coefficient.

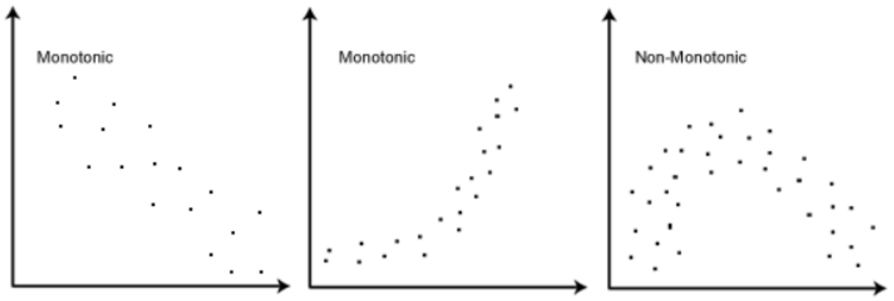


Figure 8. Monotonic and non-monotonic relationships. Source: [34]

The way of calculating the Spearman's correlation coefficient is done following this expression:

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

Being d_i the distance between the ranks of x_i and y_i , and n the number of observations (same one for x and y).

The distance is calculated according to the following process[34]:

1. Both x and y observations are ordered according to their value from the highest to the lowest one.
2. Two New columns (Rank x_i $and\ Rank\ y_i$) are generated containing values from 1 to n, depending on the value of each observation.
3. Now every observation i for both x and y has an assigned rank.
4. The next column will hold the differences between Rank x_i $and\ Rank\ y_i$

x_i	y_i	**Rank** x_i	**Rank** y_i	d_i	d_i^2
60	0	2	6	4	16
43	11	5	3	-2	4
87	9	1	4	3	9
51	5	4	5	1	1
53	13	3	2	-1	1
27	22	6	1	-5	25
					56

$$\rho = 1 - \frac{6 * 56}{6(6^2 - 1)} = -0{,}6$$

A3. Kendall's rank coefficient

The Kendall's Tau (τ) or Kendall's rank coefficient is another way of assessing the distance between two variables based on the rank of the data, as the Spearman's coefficient. This coefficient will have a value in between the rank of ± 1.

It is mostly used when any of the assumptions for the Pearson's coefficient are not met or when the sample size is too small and has too many tied ranks that could affect the Spearman's coefficient result.

The process is very similar to that of the Spearman's test, as described hereafter [35]:

1. The x variables are ordered in ascendant or descendent order. It doesn't matter which one to choose but it will be important to assess the number of discordant or concordant pairs present.
2. The y variables are ordered matching their corresponding x_i observation.
3. For each pair of observations their concordance is defined, being concordant observations those that are consistent with the chosen ordering pattern ($x_{i+1} - x_i$ and $y_{i+1} - y_i$ have the same sign) or disconcordant if there is no consistency with the order of both variables (($x_{i+1} - x_i$ and $y_{i+1} - y_i$ have different signs).

The following expression is used for computing the Kendall's Tau value:

$$\tau = \frac{N_c - N_n}{\frac{N(N-1)}{2}}$$

Being N_c and N_n the total number of concordant and disconcordant pairs, respectively, and N the total number of available observations.
The following example is provided to illustrate this process:

x_i	y_i	**Ranked** x_i	**Rank** y_i	**C**	**NC**
60	0	87	9	2	3
43	11	60	0	0	4
87	9	53	13	2	1
51	5	51	5	0	2
53	13	43	11	0	1
27	22	27	22	0	0
				4	11

$$\tau = \frac{4 - 11}{\frac{6(6-1)}{2}} = -0{,}467$$

A4. ANOVA correlation coefficient

The ANOVA (Analysis Of Variance) method gives a metric on the distance between the means of two or more groups, different from each other[36]. In order to do so, it uses the F-test, a probability distribution usually present in the analysis of variance. It can be calculated as the ratio of two Chi-squared distributions divided by their degrees of freedom:

$$F = \frac{\chi_1^2 / n_1 - 1}{\chi_2^2 / n_2 - 1} = \frac{\frac{(n_1 - 1) * {S_1}^2}{{V_1}^2} / n_1 - 1}{\frac{(n_2 - 1) * {S_2}^2}{{V_2}^2} / n_2 - 1} = \frac{{S_1}^2 / {V_1}^2}{{S_2}^2 / {V_2}^2} = (H_{F-0} = two\ variances\ are\ equal) = \frac{{S_1}^2}{{S_2}^2}$$

ANOVA test assesses variation between groups by analysing significant differences between groups (x and y) and assumes Hypothesis as H_0 – Means of all groups are equal – and H_1 – At least one mean is different- .

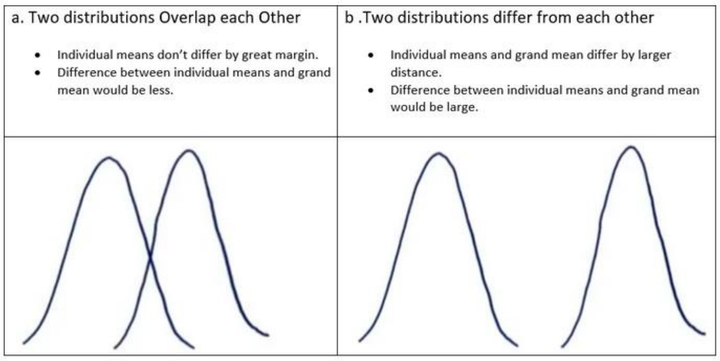

a. Two distributions Overlap each Other
Individual means don't differ by great margin.
Difference between individual means and grand mean would be less.
b .Two distributions differ from each other
Individual means and grand mean differ by larger distance.
Difference between individual means and grand mean would be large.

Figure 9. Mean distribution behaviour. Source: [36]

As pointed out in [36], this study of variability between groups is compared to within-group variability. The F-ratio will return a value close to 0 whenever difference between groups with equal variance is insignificant.

In order to decide whether a variable should be included or not for the model training, it is necessary to calculate the F-score, with the variance between groups and the variance within the groups:

$$F - Test = \frac{\frac{SumOfSquares\ Between\ Groups}{Degrees\ Of\ Freedom\ Between\ Groups}}{\frac{SumOfSquares\ Within\ Groups}{Degrees\ Of\ Freedom\ Within\ Groups}}$$

Once this is done, the computed F-value – F-calc – will be compared with the F-value from the F- tables – Known as the F-critical- (depends on the significance level α and both degrees of freedom).

If the F-calc is greater that F-critical, it will fall into the reject region, which means that the Null Hypothesis (H_0) is rejected, and there is variance between means of both groups. In this case, x has an impact on the behaviour of y, and should, therefore, be used as an input for the model.

A5. Chi-Squared test

The Chi-squared (χ^2) test is used to indicated how much distributions of categorical values (summarized in a contingency table) differ respectively to the expected values, and is computed with the following expression:

$$\chi_k^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

Where k notes the degrees of freedom (Sample size -1), O_i the observed value(s) and E_i the expected value(s).

Based on the observed data from the contingency table, the expected values are (probabilistically) obtained for each group after which the χ^2 value is calculated. In order to assess whether both variables are related, and the independent variable can be used as an input for the model, the calculated χ^2 must be compared to the critical χ^2 value, obtained from the Chi-square statistical tables (having predefined the degrees of freedom and p-value for rejecting the null Hypothesis) [37].

If the obtained value is lower than the critical value for χ^2 , then the null hypothesis (both expected and observed values are independent) is accepted and the independent variable (x) can not be used in the model.

An example if this process is provided in [37].

A6. Mutual Information

Mutual information is the given name for the Information Gain method when applied to feature selection and it is used to measure the level of uncertainty for one variable given a known value for another variable. It can be used with both numerical or categorical information. In other words, it measures the Entropy of x with respect to variables y, being the Entropy the amount of information contained in a random variable or its probability distribution. The more balanced the probability distribution is, the higher the entropy.[38]

Mutual information between two variables (I(X;Y)) is obtained by subtracting the conditional entropy for X being Y (H(X|Y)), to the entropy of X (H(X)).

$$\mathrm{I(x;y)} = \mathrm{H(x)} - \mathrm{H(x|y)} = \mathrm{I(y;x)}$$

The resulting measure is symmetrical, which means that both variables contain the same amount of information about the other one as this method assess the mutual dependence of two random variables.

In order to select those variables that are most relevant for our prediction model, we must choose the ones that result in higher values of Mutual Information with the output variable, as a null value indicates a total independence of the considered variables.

B. Wrapper-based FS algorithms

Wrapper methods are used to measure the fitness of certain features based on how good or bad does the model perform after being trained with them and the level of generalization it can achieve.

Therefore, these methods are focused on solving the real problem from the beginning (optimizing the loss function), rather than obtaining intrinsic information from each variable in order to classify their relevance with respect to the output variable.

These methods are based in iterative processes in which different combinations of variables are prepared, evaluated and modified to come up with the optimal feature combination that best fits the prediction problem. This requires a higher computational power, especially when dealing with large datasets, and need to be performed for every different model as the results obtained for a certain model can not be transferred to the next one, even if they are expected to deal with the same features (inputs and output).

The following is a brief description of the most widely used wrapper methods, together with explanatory steps to be followed during their implementation and execution.

B1. Genetic Algorithms

There are multiple applications of Genetic Algorithms (GA) in Machine Learning, being one of them the ability to select the correct features for the predictive model, allowing the best solution to be obtained from all the previous best solutions. In addition to this, GA are used to fine tune Network hyper parameters such as the ideal number of layers, units per layer, etc., as we will see later on.

GA are mathematical algorithms inspired by Darwin’s theory of Natural Selection and are included in the so-called Evolutionary Algorithms. This theory is backed up by the idea that only the fittest individuals of a society will prevail during generations.

The optimization process is done by allowing a population of individuals to evolve by randomly subjecting them to actions similar to those that act in biological evolution (genetic mutations and recombinations). This combination is aimed at maximizing the loss function (or any other predefined fitness function), and the iterative process will go on until either a threshold is reached or it has surpassed the maximum number of iterations, according to which it is decided which are the most adapted individuals, who survive, and which are the least adapted, who are discarded [39].

In order to do so, the process is generally divided into 4 phases:

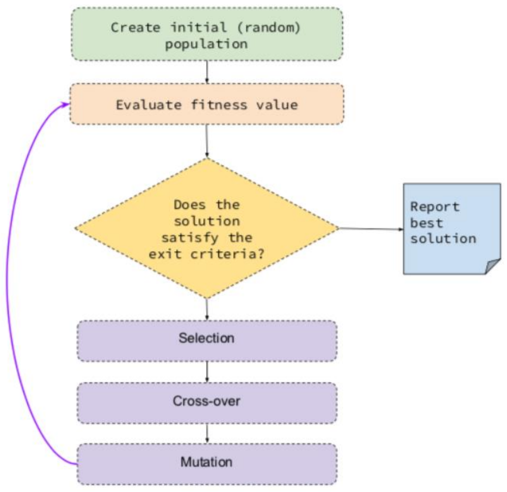

Create initial (random) population
Evaluate fitness value
Does the solution satisfy the exit criteria?
Report best solution
Selection
Cross-over
Mutation

Figure 10. GA process. Source:[40]

Phase 1. Definition the initial population – Initialization

To begin with, it is necessary to start the process by choosing a limited set of individuals, called population, that will contain a random number of possible solutions to the problem we want to solve. Most of the times, this population will be generated randomly, unless we have confidence enough to manually do so (prior knowledge required). Each individual will have a number of attributes (variables), called genes, which are joined together forming a Chromosome or individual (a solution).

A highly recommendable aspect to keep in mind is that this population is the breeding ground of our future predictors. This means that it is required to have a large population that can comprise different solutions as this will enable to explore multiple alternative paths during the execution of the model.

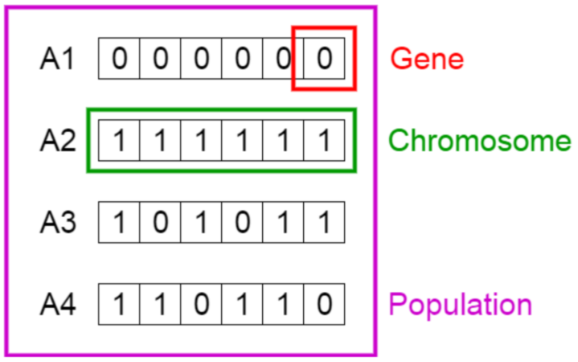


Figure 11. Randomly proposed initial solution. Source:[40], [41]

Phase 2. Definition of the fitness function

The fitness function gives a metric for each individual, based on how close each one of them is to the optimum solution of the problem.

There are different fitness functions depending on the problem that needs to be addressed, and it is the most critical and complex part when deploying GA as there is not a guidebook on which specific fitness functions use for each particular problem. In time series forecasting, for instance, the fitness function is the same as the one that computes the loss score. Some of the most used loss functions are depicted in the table containing different forecasting applications in *1.2*.

The reason why we need to use this fitness function is because the GA optimization cannot be done without having previously obtained the results of the training set for the prediction model, that is, both GA and prediction processes must be done within the same iteration. This ensures that the individual's selection will be measured based on the distance between predictions ($\hat{y}$) and the true target values (y).

When multiple objectives want to be targeted, it is possible to define different fitness functions and obtain the fittest individuals for each one of them, returning a series of optimal solutions equally optimal. This is called the Pareto frontier [39]. A decider is used to reduce the number of solutions to the required one, doing so by means of analyzing the context or predefined requests.

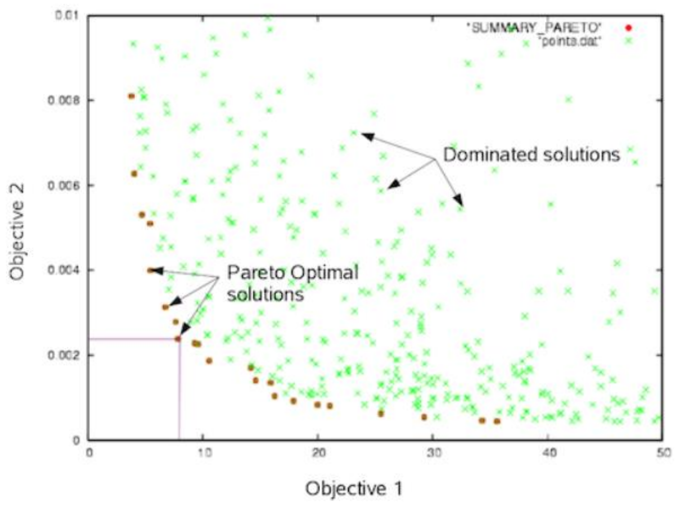


Figure 12. Pareto frontier. Source:[39]

<u>Phase 3. Genetic operators</u>

- **Selection**

After having calculated the fitness of each individual, the evolution process begins by choosing the fittest individuals from the initial population (usually two individuals are chosen). In Time Series forecasting, those who achieve a lower loss score in the first iteration are selected for reproduction.

- **Cross-over**

Once we are left with the best solutions possible (parents), it is time to create the new generations of individuals, based on their genes, among which will be chosen again the fittest ones to move on to the next iteration.

The cross-over point between parents is randomly chosen in between their genes, after what offsprings are "born" containing a mix of their parents' genes. The population size of the new generation (offsprings and parents) should match the dimension of the initial population.

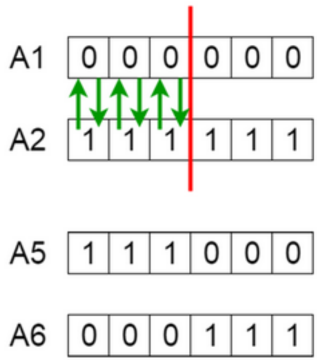


Figure 13. New solutions (A5,A6) obtained from initial population parents. Source: [41]

- **Mutation**

There is a chance that, due to the low randomness of the cross-over, we might get a limited new population that will lead to an early convergence of the problem after getting stuck in a local optimum.

In order to avoid this, a low probability shuffle mutation is allowed, which results in a much more diverse population as offsprings don't identically mirror sets of their parent's genes. This is an analogous step of biological evolution and enables the GA to try more complex combinations that wouldn't emerge from simply crossing-over genes.

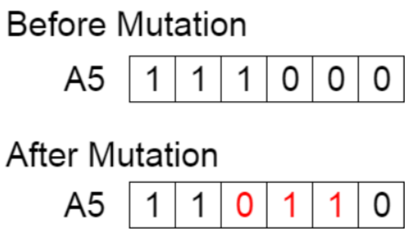


Figure 14. Before and after mutation of A5 individual. Source: [41]

<u>Phase 4. Termination</u>

As this iterative process can not go on forever, there are three ways of stopping it.

a. Global minimum has been reached and no further optimization can be done, or a performance threshold has been reached.
b. The process has reached a maximum number of iterations or runtime.
c. If the population converges, the offsprings genes will not vary significantly from their parents'. If this point is reached, we must choose a solution from the generated ones.

B2. Harmony search

As it has been mentioned before, wrapper methods tend to have a much higher computational weight when compared to alternative FS methods. To this end, the Harmony Search (HS) method enables to reduce the amount of information processed by the model and number of iterations required as it has two main differences compared to GA [3].

1. Only one solution is generated per iteration. This leverages computational weight as only one solution (and not a whole population) is evaluated.
2. Initial population is selected probabilistically.

This model was first proposed in 2001 and was inspired by the way in which musicians improvise a harmony. In order to come up with the best possible combination, musicians try different combinations of pitches they know from experience (memory) and adjust the pitch of each instrument until they obtain the harmony they were looking for. This same procedure is mimicked by the Harmony Search algorithm to try to come up with the optimal solution for a given problem.

Same as in GA, this method has 4 main steps that structure its inner working:

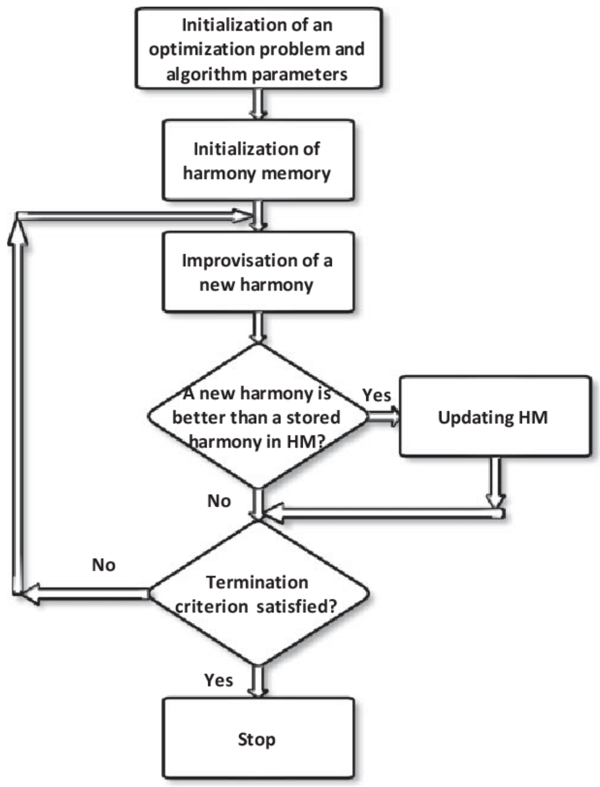

Initialization of an optimization problem and algorithm parameters
Initialization of harmony memory
Improvisation of a new harmony
A new harmony is better than a stored harmony in HM?
Yes
Updating HM
No
No
Termination criterion satisfied?
Yes
Stop

Figure 15. Harmony Search method. Source:[42]

Phase 1. Initialization of HS parameters and Harmony Memory (HM)

The first thing to do when working with HS as a FS method, is to initialize the problem by generating a random number of N harmonies (solutions to the problem) that will be stored in the Harmony Memory. These harmonies will contain d values, corresponding to the number of variables that que initially have or that we want to pass on to the model.

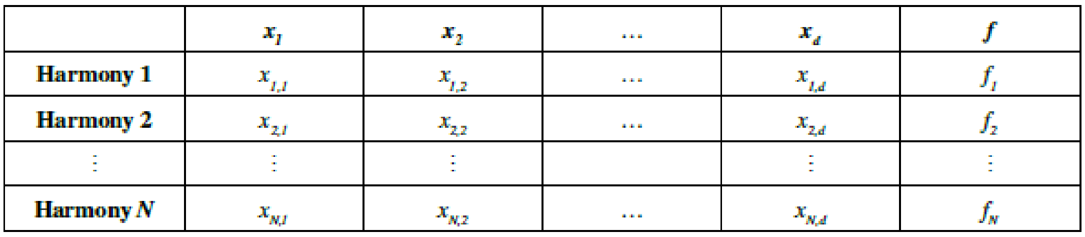
A way of generating these harmonies can be done by means of the following expression, proposed by [42]:

$$x_{i,j} = l_j + rand\left(u_i - l_j\right) \qquad i = 1,2,3,\ldots,N \qquad j = 1,2,3,\ldots,d$$

Being, u_j and l_j the upper and lower bounds for variable j, and $rand$ a randomly generated number with a uniform distribution [0,1].

After doing so, each harmony is evaluated within the predictive model and receives a value for the loss function (f_i). Once again, the loss function that be used to compute the "fitness" of each harmony will be the optimization function of the predictive algorithm, just like it was explained in the above section.

All this information will be stored in the HM in the form of a matrix, as shown in Figure 16.



	x_1	x_2	…	x_d	f
Harmony 1	$x_{1,1}$	$x_{1,2}$	…	$x_{1,d}$	f_1
Harmony 2	$x_{2,1}$	$x_{2,2}$	…	$x_{2,d}$	f_2
⋮	⋮	⋮		⋮	⋮
Harmony *N*	$x_{N,1}$	$x_{N,2}$	…	$x_{N,d}$	f_N

Figure 16. HM structure matrix. Source: [42]

<u>Phase 2. Improvisation of a new harmony. HS operators</u>

Once all the harmonies are stored, together with their corresponding loss function values (loss score), it is time to begin the iterative process that will engage steps 2 and 3 until an optimum is reached. In this case, and similar to the Phase 3 of GA, the process of generating a new harmony has two main steps [42].

- Harmony Memory Considering Rate

This step comprises the improvisation of a new harmony $x_{new} = \left(x_{new,1,\ldots,}x_{new,d}\right)$ that will be evaluated within the prediction model and compared to the stored harmonies in the HM. This new harmony will be obtained using all the stored harmonies generated in Phase 1.

There are two ways of generating a new harmony; if $rand$ (a randomly generated number [0,1] following a normal distribution) > HMCR (Harmony Memory Considering Rate, [0,1]), then $x_{new,j} = l_j + rand\left(u_i - l_j\right)$; else if $rand \leq$ HMCR , one of the stored harmonies will be randomly selected $x_{new,j} = x_{k,j}$, where 1≤ $k \leq N$.

HMCR is one of the main operators of Harmony search, just as cross-over and mutation where in GA, and it is defined as the probability of choosing a component of the Harmony Memory [43]. Higher values of HMCR imply that the sound will be closely related to those stored in HM, and lower values mean that there is a small probability of generating a new sound from the possible range.

- Pitch Adjusting Rate

This step is closely related to the mutation step of GA as it is used to scape local optima that would cause the algorithm to early converge after becoming stuck.

In this case, Pith Adjusting Rate (PAR) provides a means of mutating the obtained harmony to a close value, following a low probability distribution, and it represents the probability of a HM candidate to be modified, acquiring a value in the range of [0,1]. The higher the PAR value, the higher the chance of pitch adjusting a harmony.

In order to Pitch adjust his new harmony, the PAR value will be compared again to a randomly generated number from 0 to 1.

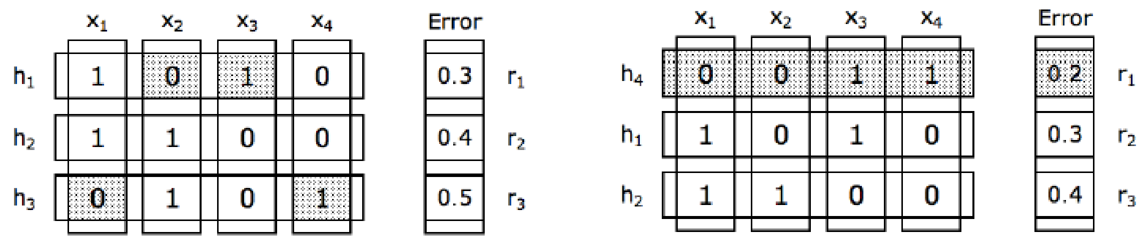
If $rand \leq$ PAR, the newly generated harmony will be modified according to the following expression; $x_{new,j} = x_{new,j} + bw * (rand - 0,5) * |u_i - l_j|$, being bw the bandwidth of generation or stepsize (distance between the new harmony and HM values). Else if $rand >$ PAR, the new harmony will remain the same.

Phase 3. Replacement

Once the new harmony has been fully acquired, we are left with a new feasible solution to our problem. Therefore, this new harmony is evaluated in the prediction model and its loss score computed.

Once this is done, this harmony and its corresponding loss score is compared to the worse harmony stored in the HM, that is, the one with the worst loss score.

If f_{new} is better that f_i, harmony i will be removed and replaced by the new one. Otherwise, the new harmony will be dismissed, and a new harmony will be generated following the process of Phase 2.



	x_1	x_2	x_3	x_4	Error	
h_1	1	0	1	0	0.3	r_1
h_2	1	1	0	0	0.4	r_2
h_3	0	1	0	1	0.5	r_3

	x_1	x_2	x_3	x_4	Error	
h_4	0	0	1	1	0.2	r_1
h_1	1	0	1	0	0.3	r_2
h_2	1	1	0	0	0.4	r_3

Figure 17. HM before and after replacement h3-h4. Source:[3]

Phase 4. Termination

The stopping criteria oh the HS algorithm closely resembles that of GA as the process can be stopped whenever a maximum number of iterations is reached, or the global optimum found.

B3. Temporal Memory search

Other authors propose alternative methods to reduce the computational weight that characterizes wrapper -based methods. One of these methods is the Temporal Memory Search [3].

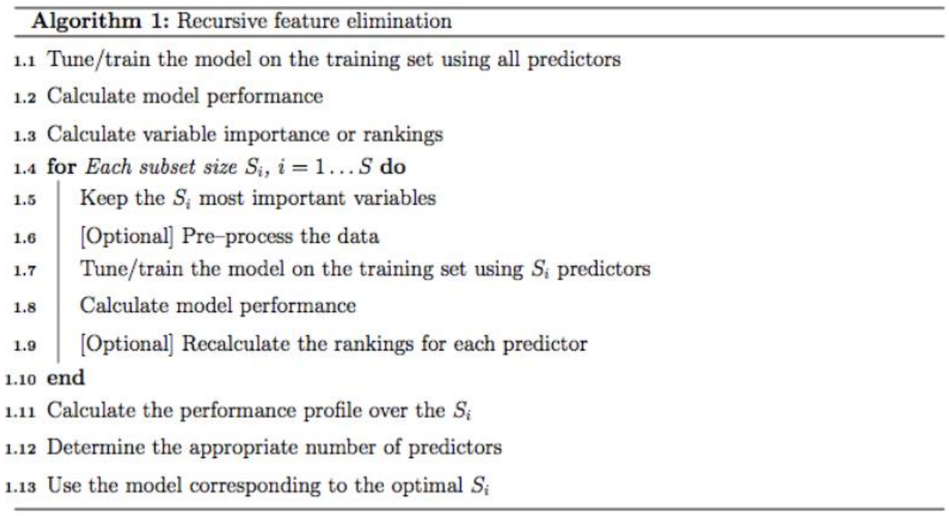
This method tries to identify the mount of temporary memory needed to solve the issue, being the memory the past time values (lags) of the time series.

Although this seems to be a useful tool to optimally select the most relevant time lags of each series, its deployment is still very limited and few information can be found about it.

B4. Recursive Feature Elimination

Recursive Feature Elimination (RFE), as its name states, is a means of feature selection based on an iterative process in which an initial set of variables is trimmed gradually until only the most relevant variables are left, or until the desired number of relevant features is achieved.

In each iteration, each feature is ranked according to a coefficient or feature importance attribute, after which, those with the lowest importance are eliminated. Then, the model is trained again, and the left variables are given a new feature importance attribute. The “score” is given by a supervised learning estimator, for example linear models would be used to estimate weight coefficients and tree-based algorithms would assess each feature importance within the model.



```
Algorithm 1: Recursive feature elimination
Tune/train the model on the training set using all predictors
Calculate model performance
Calculate variable importance or rankings
for Each subset size $S_i$, $i = 1 \dots S$ do
    Keep the $S_i$ most important variables
    [Optional] Pre–process the data
    Tune/train the model on the training set using $S_i$ predictors
    Calculate model performance
    [Optional] Recalculate the rankings for each predictor
end
Calculate the performance profile over the $S_i$
Determine the appropriate number of predictors
Use the model corresponding to the optimal $S_i$
```

Figure 18. RFE pseudo-code. Source:[44]

In order to determine the correct number of optimal features to be left, this number can be either fixed as a parameter for the RFE algorithm or determined via cross-correlation. In this case, cross-correlation will enable to evaluate different variables subsets that will help to select the correct number of features after N iterations.

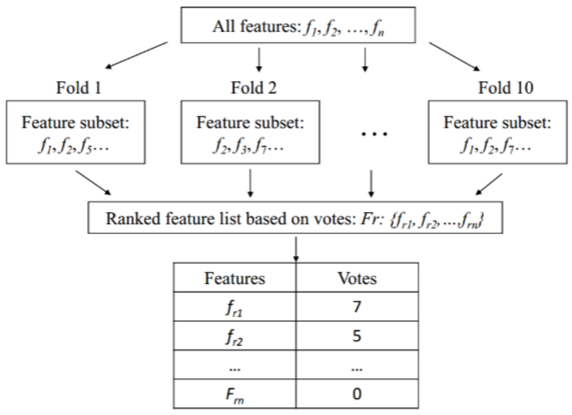

All features: $f_1, f_2, \ldots, f_n$
Fold 1
Fold 2
Fold 10
Feature subset: $f_1, f_2, f_5 \cdots$
Feature subset: $f_2, f_3, f_7 \cdots$
Feature subset: $f_1, f_2, f_7 \cdots$
Ranked feature list based on votes: Fr: $\{f_{r1}, f_{r2}, \ldots, f_{rn}\}$
Features
Votes
f_{r1} 7
f_{r2} 5
... ...
F_{rn} 0

Figure 19. Feature selection strategy using cross-validation. Source: [45]

Before executing this method, some tuning parameters must be adjusted such as [46]:

a) (Minimum) Number of features to be selected at the end of the process
b) Step: number of features pruned after each iteration
c) The estimator that will be used to evaluate each feature
d) Verbose, in order to control verbosity of output
e) Number of folds (in case of using CV-RFE)

An example of this method is provided in [44].

B5. Sequential Feature Selection

This Feature selection algorithm works in the opposite way compared to RFE.

In this case, an initial empty feature set is competed after each iteration with a number of variables until there is no improve in accuracy for the trained model.

There are 2 main types of Sequential Feature Selection (SFS) algorithms[47], floating or non-floating. The main difference between these two models is that floating algorithms include an additional step to include or exclude variables once excluded or included, respectively, which allows to analyze a higher amount of subset combinations. This is an optional step that only succeeds if the removal or addition of a particular feature proves to improve accuracy, otherwise, this step will be skipped.

Also, there are 2 types of feature search, depending if the feature selection is done backwards or forward. This means we can distinguish 4 types of algorithms within SFS:

- Sequential Forward Selection – SFFS
- Sequential Backward Selection – SBFS
- Sequential Forward Floating Selection – SFFFS
- Sequential Backward Floating Selection – SBFFS

All this algorithms follow a similar process in order to perform an optimal feature selection, as described in [47]:

<u>Phase 1. Initialization</u>

- SFFS & SFFFS
  - The algorithm is initialized with an empty set of variables X_k=$\emptyset$ and $k = 0$
- SBFS & SBFFS
  - The algorithm is initialized with all the available features $X_k = Y = \{y_1, y_2, \dots, d\ \}$ and $k = d$

<u>Phase 2. Inclusion/exclusion of variables</u>

- SFFS & SFFFS -Inclusion
  - An additional feature $x^+$is transferred from the original set Y into X_k, being x^+ a feature that achieves the best performance for the prediction model when trained with X_{k+1}

$$X_{k+1} = X_k + x^+ \qquad k = k + 1$$

- SBFS & SBFFS -Exclusion
  - In backward selection models, a x^- feature is removed from the initialization set X_k, being x^- the feature that improves model accuracy when trained with X_{k-1}

$$X_{k-1} = X_k - x^- \qquad k = k - 1$$

<u>Phase 3. Conditional Inclusion/exclusion. Only for SFFFS and SBFFS</u>

- SFFFS – Conditional Exclusion
  - After having added a variable into X_k, we evaluate again the complete set of X_k and exclude any variable (x^-) that would lead to an improvement of the model.
- SBFFS – Conditional Inclusion
  - In this case, we look for any other variable x^+ that could be added back (in case it was removed previously) in order to achieve better accuracy when training the model.

<u>Phase 4. Termination</u>

After several iterations, this process would return a final feature subset X_k, being $k < d$, of optimal solutions with which achieve the best possible results with the predictive model.

The stopping criteria can be either done manually by predefining de desired number of features to be stored in X_k, or automatically whenever the improvement ratio remains constant.

C. Embedded FS algorithms

Embedded FS algorithms use algorithms to penalize features with coefficients too high in order to reduce complexity and avoid over-fitting or variance of a model by adding extra bias. These algorithms try to take advantage of the benefits of both filter and wrapper methods.

The idea behind embedded FS methods is that, if a model is evaluated with a very large number of variables, the coefficients associated to each variable will increase, which makes it complicated to choose those that are really important for the model to behave correctly.

Therefore, regularized methods, such as L1 regularization (LASSO), L2 regularization (Ridge Regression) or decision tress, are used to control the size of features' weights [48].

Given that these methods used their own models, it is complicated to adapt their use in the NN's domain.

3.3.2 Hybrid Models

NNs' different architectures aim to provide a wide range of options in order to target issues of multiple natures, such as image/video classification, forecasting, speech recognition, natural Language Processing (NLP), and many more (see Section CHAPTER 2.).

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

Some of these models will be listed and explained hereafter, paying special attention to their advantages compared to simple architectures.

Deep and Wide Neural Networks (DWNN)

DWNNs' are a new model proposed in [2], where a Convolutional layer is added to the hidden state of a Recurrent Neural Network. This way, the model not only accounts for the depth provided by RNNs' (time dimension, in the form of number of time lags), but also the width associated to CNNs' (variable number of data sets).

What this combination does in the end, is provide NN models with an additional tool to identify and locate relevant relationships between temporal input data sets. This is very useful as m independent RNN models would be required to deal with problems involving m sets of related temporal data, and even by doing this, the model would neglect correlation between each sequence. By adding a CNN layer between $t-1$ and t in the hidden state, being the input of this layer the concatenated hidden states of all m RNN models at time $t-1$, and its output the state input for the RNN at t, we are able to merge all RNN models and retain the memory of the previous lags of the RNN.

This CNN layer is formed by convolution layers, a pooling layer and a full connection one in order to adapt the output into the required shape of a hidden state.

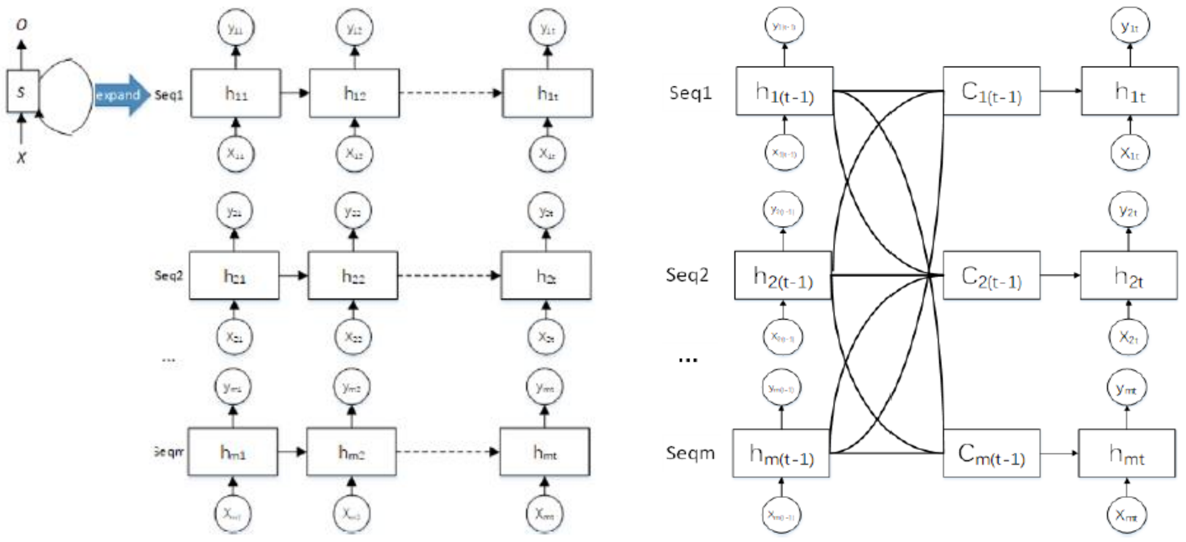


Figure 20. Simple RNN mode VS DWNN model. Source: [2]

Another variation to this would be to add CNN layers at every k-th time step. This would reduce the risk of overfitting of the training model, as well as simplify the processed information, while maintaining the ability to extract correlation. These models would also be advantageous when dealing with sequences with different periods. This is, in case we had Seq1 with periodicity equal to 1, Seq2 equal to 3 and Seq3 equal to 6, we would place the CNN layer at the 6-th time step, or any multiple of 6 to ensure that all these behaviors are captured.

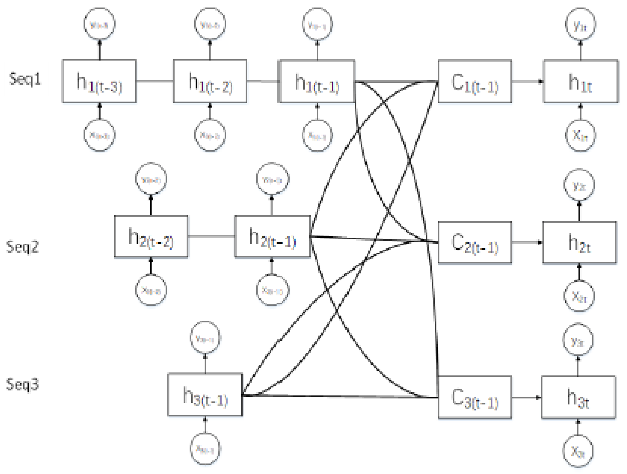


Figure 21. DWNN with different periods. Source: [2]

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

<u>CNN + LSTM – TreNets</u>

Introduced in [49], TreNets are hybrid Neural Networks architectures for the trend prediction of time series. This hybrid architecture combines long short-term memory (LSTM), a convolutional neural network (CNN), and a feature fusion layer.

In this case, the LSTM network contains and passes on the historical trends that contain the long-term contextual information of the time series. This historic information is relevant as it may naturally affect the trend evolution. Alternatively, raw local data serves as an input for the CNN, which will extract useful information about the local behavior of the time series in order to determine the dependency of the current trend and pattern transition point. This kind of information can be of great relevance when predicting abruptly changing trends. The following image, extracted from [49], might help to understand the need for such local data.

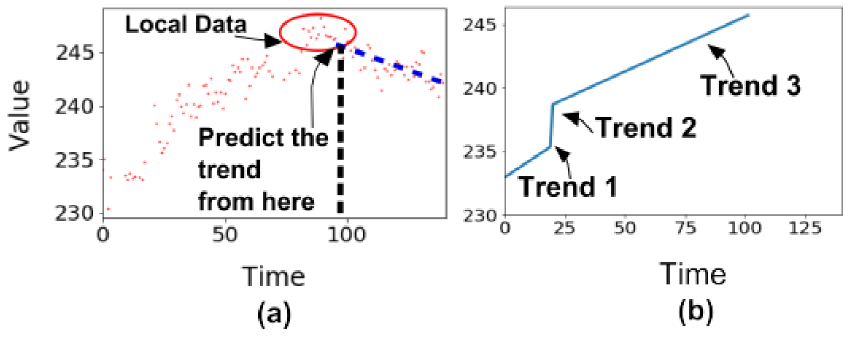


Figure 22.. Local and historic trend data. Source:[49]

As it can be observed, the historical trend data (Trends 1- 3) would, most likely, indicate an increasing trend as well for the following period. However, Local data, shown in (a), indicates a deceleration of this behavior or even a change in the pattern, as it is confirmed by the time series vales from t=100 onwards.

The information extracted from both networks (LSTM (-L) and CNN-(T)) is then fed to a feature fusion layer that will join both output representations into a single joint feature. This joint feature will then be used by the output layer to forecast the following trend. The mathematical expression corresponding to this prediction can be found in [49].

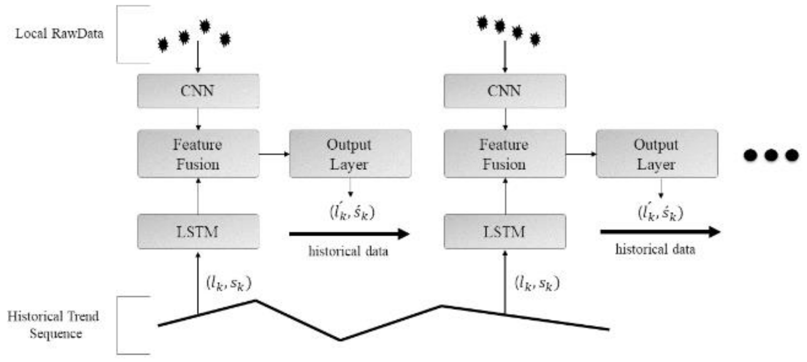


Figure 23. TreNet schematic representation. Source: [50]

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

<u>Bi-Directional LSTM + CNN layer</u>

Modelled in [20], a model combining both bi-directional LSTM networks and CNN layers, seeks to achieve a higher degree of accuracy while reducing the probability of overfitting.

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

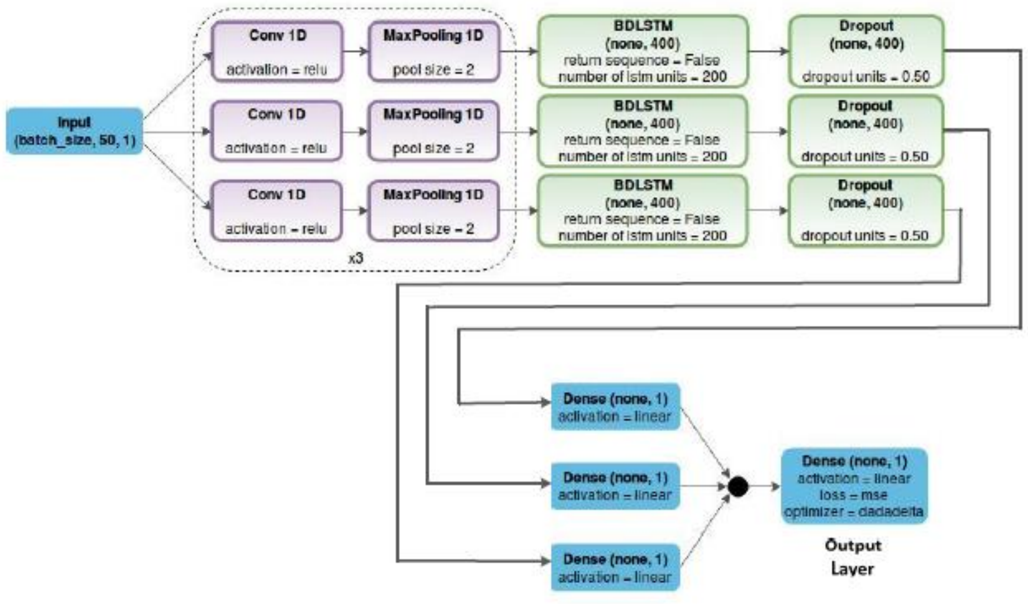


Figure 24. Proposed hybrid architecture CNN- BI LSTM. Source:[20]:

Figure 24 shows that the proposed model consists of three separate layers, each one taking the entire data sequence as input and is made up of 3 separated groups.

The first group composing each pipeline contains three 1-D CNN with ReLU as activation function and a Max pooling layer.

The next group is made up of two bi directional LSTM layers (one for each forward and backward passes) whose output is concatenated in order to obtain a single output that can be fed into the next group. In this case, a 50% dropout layer is used in order to prevent problems derived from vanishing gradients. Output from this group is then fed into a dense layer with one unit and Linear activation function, returning the output of each pipeline.

Finally, all three outputs are concatenated and passed through a dense layer that generates the final numeric value for the predictions.

This model claims to have increased accuracy by 9% with regards to a single pipeline CNN- Bi LSTM model [20].

Low-High CNN

Described in [27], Low-High CNNs are a novel architecture that combines multi CNNs with multistep Attention modules.

As it has been previously explained, CNNs provide and advantageous feature selection mechanism when dealing with long span data series, as are able to extract information without being corrupted by local variations.

In this case, various sets of CNNs are used for extracting (1) Low Level features and (2) High level features. Low level features are made up of local behaviors in different time steps (curves, onwards or downwards slopes, etc.) whereas High level features are built on top of these and extract larger shapes in temporal data.

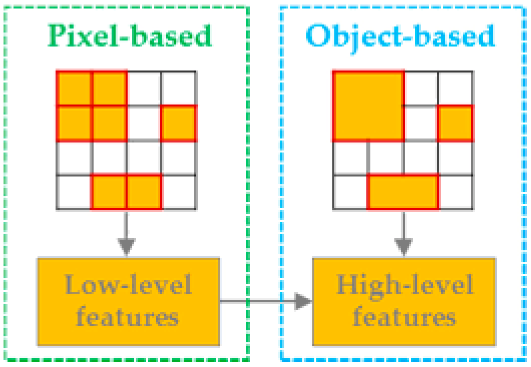


Figure 25. Low-High Level features in an image. Source:[51]

In order to combine these, a multistep attention module is used. The attention module matches both outputs by identifying relevant context, just as it would be used in NLP for sentence translation. This enables to get better understanding of the problem context and helps in assessing how different data points related to each other without losing the temporal component. In order to ensure that both Low and High level feature convolutions have a matching output, linear mapping is applied to all the high level convolution layers, to the first layer of the low-level, and to all layers before computing attention scores.

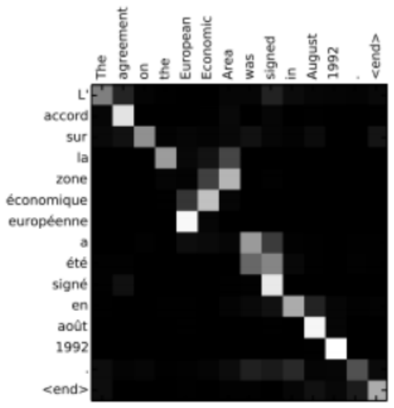


Figure 26. Attention module for NLP. Source:[52]

In addition to this, residual mapping is added to every convolution (Low and High) calculation in order to prevent vanishing or exploding gradients as the networks acquires a higher degree of abstraction after each iteration.

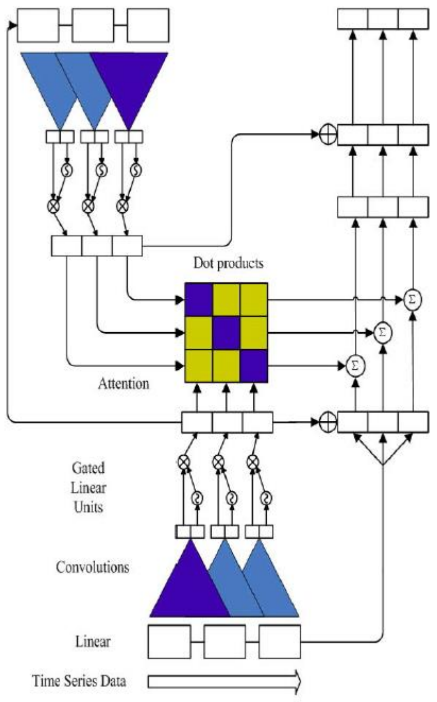


Figure 27. Overview of the proposed model architecture. Source: [27]

The above figure shows and schematic approach to the proposed model architecture. Time series data in convoluted – Triangle icons – (Bottom, Low Level Features, Top, High level features) and their outputs are consolidated by the attention module (central matrix). Conditional inputs computed by the attention (center right) are added to the high-level features which then predict the target sequence (top right). The sigmoid and multiplicative boxes illustrate Gated Linear Units.

3.3.3 Hyperparameter tuning

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

Some of the most common or important hyperparameters to adjust within a NN are [53]:

- Number of hidden layers: Increasing the number of hidden layers is usually believed to increase model accuracy.
- Cell units (neurons) per layer: Same as with the number of hidden layers, a greater number of neurons per layer might help to optimally identify relevant behavior in data, as more interactions between variables can be taken into account. Having a large number of neurons might lead to an increase in computational weigh and even overfitting.

- Parameter initialization: It is necessary to initialize the weights in the first pass. These values can be set to zero or obtained with a random function. This can lead to vanishing or exploding gradients, which reinforces the need to find a way of easily initializing them without compromising its training.
- Learning rate: It represents the amount weights are increased during training. It usually has a positive value from 0 to 1. This value is one of the most relevant ones, as it can lead to heavy and long training process after which the process could get stuck (low values) or too fast and cause the training process to become unstable and achieve sub-optimal sets of weights (large values)[54]
- Loss function: Seen in section 2.1.1, it is the function designed to calculate the distance between the predicted output and the expected output. After computing the loss score, a learning algorithm (usually Gradient Descent) is then used to update the weights in a way that might achieve a better loss score in the next iteration.
- Epochs, iterations and batch size: These three hyperparameters define the way in which data is fed to the model. As explained in [55]:
  - An epoch is a forward pass and a backward pass of *all* the data samples in the training dataset.
  - The batch size is the number of data samples in each forward or backward pass. This value is set when the number of variables is too large to be run in one single epoch, or when the complexity of the model makes it necessary.
  - The number of iterations is the number of backward and forward passes using the information contained in each batch.

  For example, if we had 200 variables with a batch size of 100, it would take 2 iterations (200/100) to complete 1 epoch.

  These values (specially the batch size) can significantly impact the model

- Dropout regularization: Defines the number of neurons not trained in each epoch in order to avoid overfitting of the model during training.
- Optimizer algorithm and momentum: The neural network optimizer is in charge of running gradient descend in order to actualize the weights of each variable. The way to do so varies from one optimizer to another which can impact the model performance.

In order to select the right values for each hyperparameter a lot of experience is required, and not even experience can guarantee optimal results. Although there is no straight-forward way to fine tune them, there are some methods that can facilitate this task and make it less complex, among which four stand out [53], [56], [57] .

A. Hand Tuning

Although this method might seem obvious and too simple, hand tuning might lead to better results when tuning a hyperparameter that other methods that will be reviewed in this section.

The reason for this is that we can easily learn from our previous mistakes, which helps to quickly adapt the model when the results improve or get worse after modifying the value of a certain hyperparameter. Nevertheless, a great inconvenient of this model is that it doesn’t work when needing to optimize several hyperparameters because, as it was explained before, a change in one hyperparameter might force to change the rest and start the process again. Another issue is that it can easily force the problem to converge in a local optimum, without providing any tools to escape from it.

In the end, this method can be used whenever knowledge and previous experience supports it, but it is not a scientific method and does not account for under optimized hyperparameters.

B. Grid Search

Grid search methods stand as the simplest way to automatically select the optimal value for a number of hyperparameters.

This method is based in an iterative process that tries multiple values for each hyperparameter. These values can be either predefined or sorted out from an interval with a fixed step size and the model will train the model for each possible value and return its associated loss score. In the end, this method turns out to be an in-depth search based on some hand tuned spectrum of numbers within the hyperparameters space of the algorithm.

When compared to hand tuning, this method allows to tune a number of hyperparameters all together but again requires some expertise to select the range of options for each hyperparameter. On the good side, Grid search methods provide a means of mapping the problem space and more optimization capability.

This method can be used with simple NN models but has a limited deployment among complex deep models. This is due to the fact that it can lead to a really reduced speed training process, depending on the number of hyperparameters to be tuned and the possible alternatives for each one, turning out to be a very inefficient method.

C. Random Search

If we were to use a grid search method to optimize the 6 abovementioned hyperparameters, trying for each one of them 10 possible values, we would need to run the model 1000000 (10^6) times. If each training process was to take up to 5 minutes to complete, the model would be able to try all the possible values in 9.5 years, without considering any further tuning in case the predefined are unable to yield any optimal results.

In order to avoid this, Random Search methods are strictly linked to Grid Search methods but only try randomized values of hyperparameters[53]. These values are gathered from the entire problem space, rather than form just promising areas that could hide a local optimum. If we were able to reduce the number of options for each hyperparameter to 5, the time required to train the model would be reduced to 0.17 years, which is a huge decrease compared to the 9,5 years required with a regular Grid Search method.

One of the main issues with this method is that it can sometimes leave some space points uncovered and it evaluates points that are too close to each other to really make a significant improvement. In order to avoid this, quasi-random sequences (also known low-discrepancy sequences) help to spread more evenly the data points. Some of these quasi-random sequences are the Sobol, Harmmersley, Halton, Kronecker and Niederrelter sequences.[56], [58]

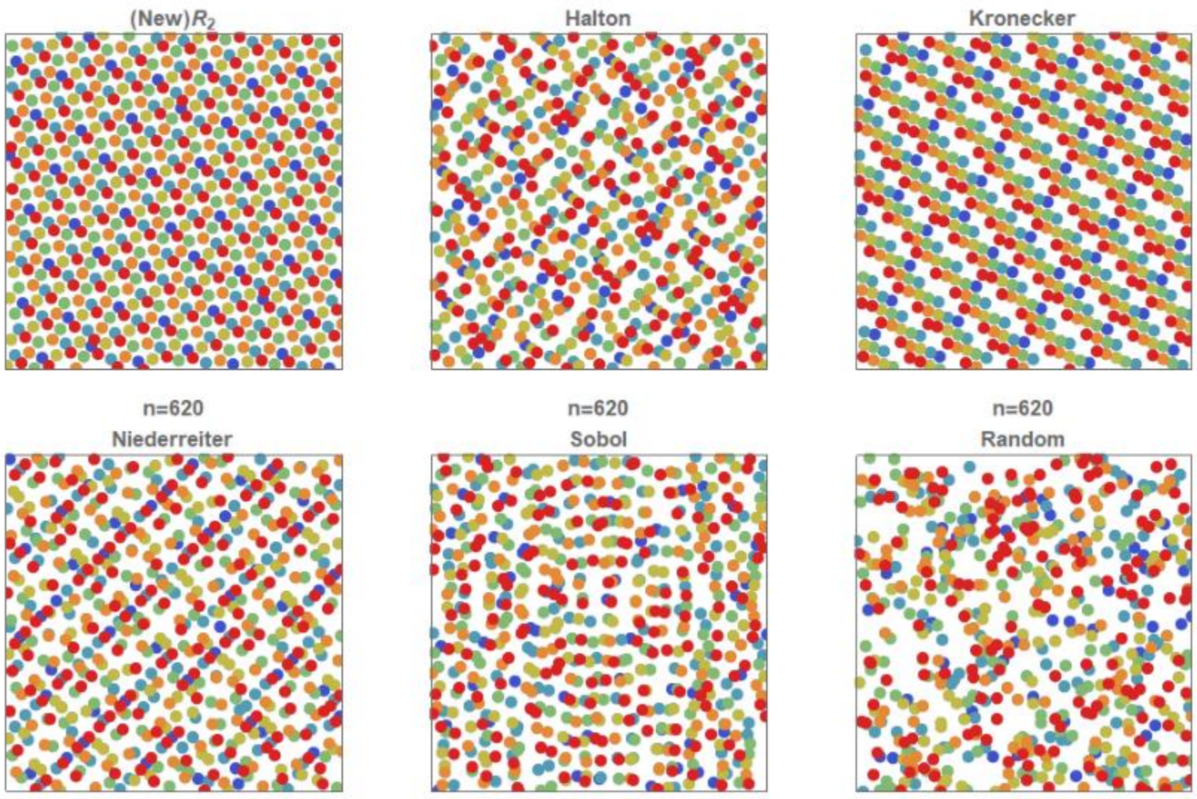


Figure 28. Comparison of some quasi-random search methods. Source: [58]

The main issue with this method is that there is still a need for expertise in order to choose the correct method or distribution based on the data observations we have and even once the results have been obtained, these might not be intuitive or difficult to improve on due to their lack of explanation.

Together with Grid Search methods, Random Search methods does not retain information about past evaluations. This means that they do not learn from past mistakes which means that a lot of time can be invested in evaluating values that will not make any improvement. As it was mentioned before, this is the reason why hand-tuning might be a better solution to these methods when the problem dimensions allow so.

D. Sequential Model-Based Optimization (SMBO) – Bayesian Optimization

Sequential model-based optimization methods represent a substantial improvement compared to Grid and Random search methods as they base their operation in trying to improve past evaluations through a probabilistic model, making assumptions about unobserved values through an Acquisition Function.

Bayesian methods have a very similar skeleton that allows some modifications depending on the used model [59]:

Step 1. Create a model that maps the hyperparameters to a score of the prediction model's loss function. This is called a surrogate for the loss function – $p(score|hyperparameters) = p(y|x)$– and it is used because it provides a much simpler way of optimizing the hyperparameters than by directly evaluating the loss function.

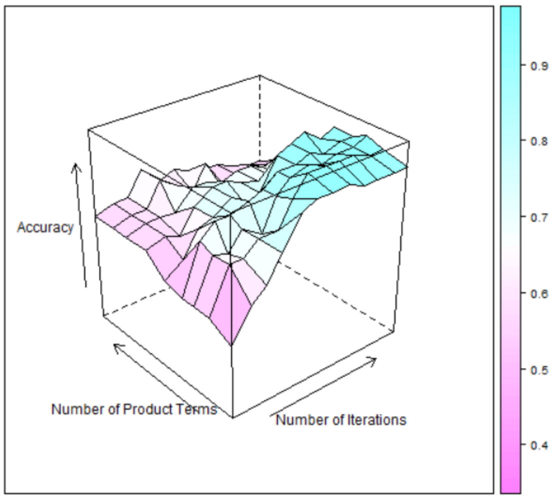


Figure 29.Surrogate Function for an automobile data set with 2 hyperparameters. Source:[60]

Among the available choices for construction of the surrogate model, Gaussian Process, Random Forest and Tree Parzen Estimators (TPE) have had the most popularity in recent developments. Further Information about these methods can be found in [56], [57], [59], [61].

Step 2. The model identifies those hyperparameters' values that perform best on the surrogate function.

Step 3. Values identified in the previous step are applied to the prediction model and evaluated with the real loss function.

Step 4. Loss score results obtained in Step 3 are updated in the surrogate that looks for the next values to evaluate. In this Step it is required to have a Selection/Acquisition Function, or criteria method, to disclose which values are more interesting to evaluate in the next iteration, being Expected Improvement the most used one as it has proven to obtain good results in different environments. Other functions are Probability of Improvement, minimizing the Conditional Entropy of the Minimizer and bandit-based criterions [61].

Expected Improvement tries to find the best hyperparameters under the surrogate function, that is, maximizing the Expected Improvement with respect to x, and is expressed as follows:

$$EI_{y^*} = \int_{-\infty}^{y^*} (y^* - y)p(y|x)\, dy$$

Being $y^*$the expected value of the loss function, y the actual value of the loss function, x the set of hyperparameters and $p(y|x)$ the surrogate function.

Step 5. Steps 1-4 are repeated through an iterative process running until the maximum number of iterations is reached.

Therefore, the basis of these methods is information. The more information the model has from previous observations, the better choice of hyperparameters' values will make in further iterations, achieving better results than the previous explained Search methods in less iterations.

E. Other methods

Although these 4 methods retrieve most of the attention, there are also alternative resources that may be used for hyperparameter optimization.
Gradient-based optimization, Evolutionary Algorithms and Population-Based Algorithms are some of these alternative methods.
Gradient-based optimization methods apply reverse stochastic descend methods with momentum to evaluate gradient descend related to each hyperparameter. Then, just as it is done with the weights modifications, values for each hyperparameter are upgraded in order to obtain a better loss score for the model.

Evolutionary Algorithms for hyperparameter tuning work just as described for Genetic Algorithms in section 1.3. These methods are widely used specifically for neural networks optimizations as they are easy to compute and perform well with various architectures [62].

Population-Based Algorithms are a combination of both parallel search methods (Grid and Random Search) with Sequential Search methods (Hand and Bayesian optimization). It jointly learns both hyperparameters and networks weights, conducting a wholesome optimization of the prediction model. It also reflects the inner workings of Evolutionary Algorithms as it mutates periodically the hyperparameter values. These mutations are based on previous non-converged observations. Hyperparameters evolve, eliminating poorly performing hyperparameters' values and replacing them by previously obtained better ones. The initial set of values must be given beforehand, but the modifications on the hyperparameters' values removes the need for further hand tuning. More information on this topic can be found in [63]

CHAPTER 3. CASE STUDY

Whereas chapter number 2 has served as an introduction to different prediction models and techniques that could be used to forecast time series values, it still remains unclear which would be most ideal to work with.

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

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

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

3.1 Software settings

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

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

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

3.2 Data Set

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

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

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

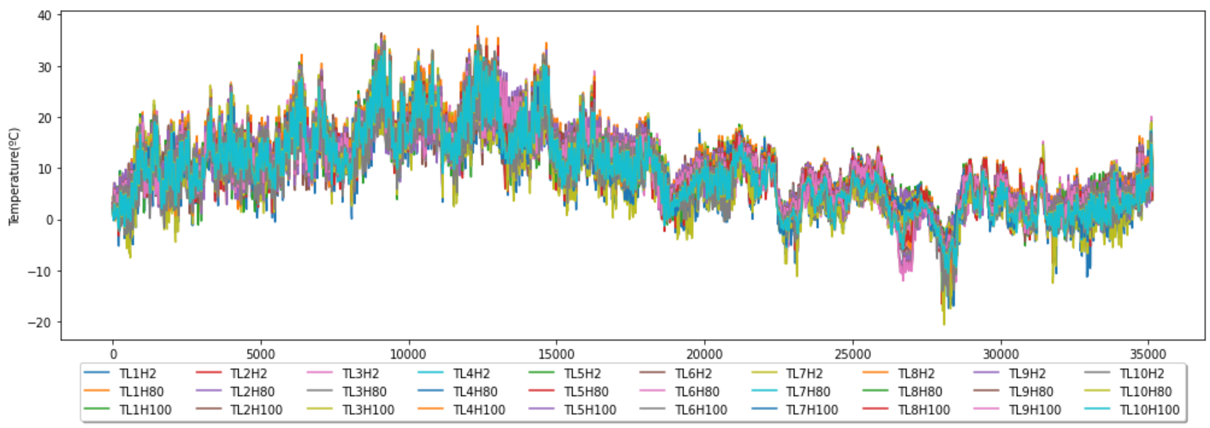


Figure 30. Temperature (ºC) variables. Min. = -18,7 ºC, Max. = 37,5 ºC

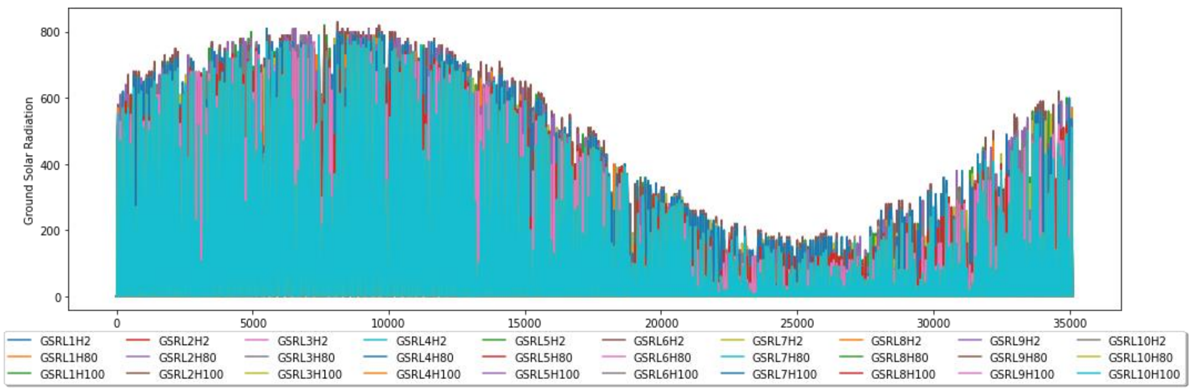


Figure 31. Ground solar radiation

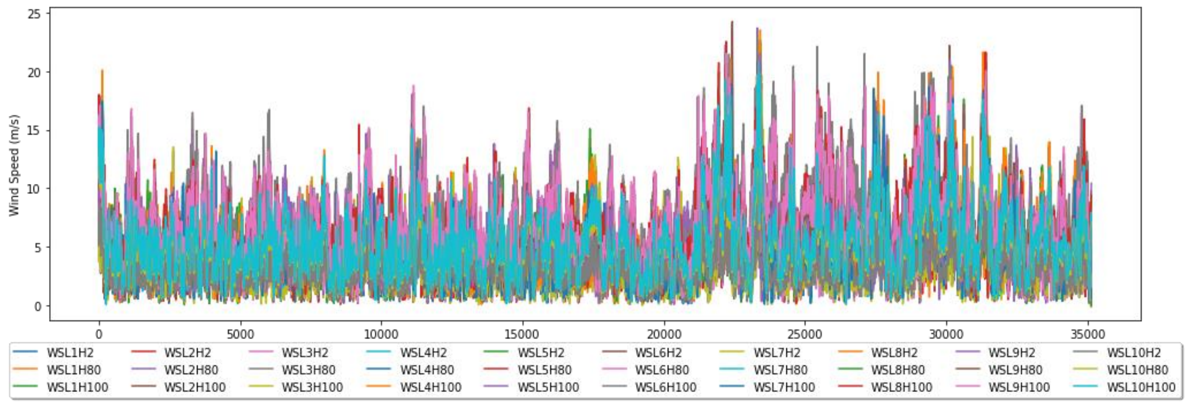


Figure 32. Wind speed (m/s)

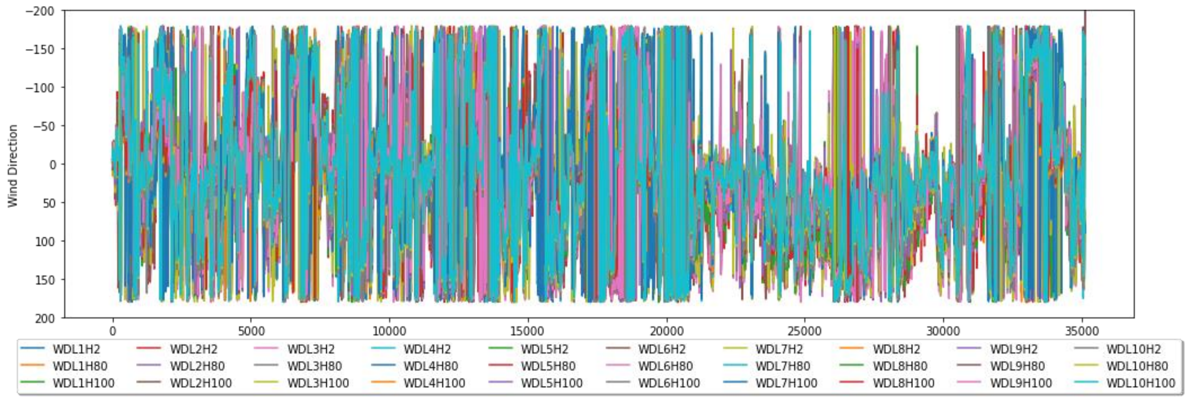


Figure 33. Wind direction (º)

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

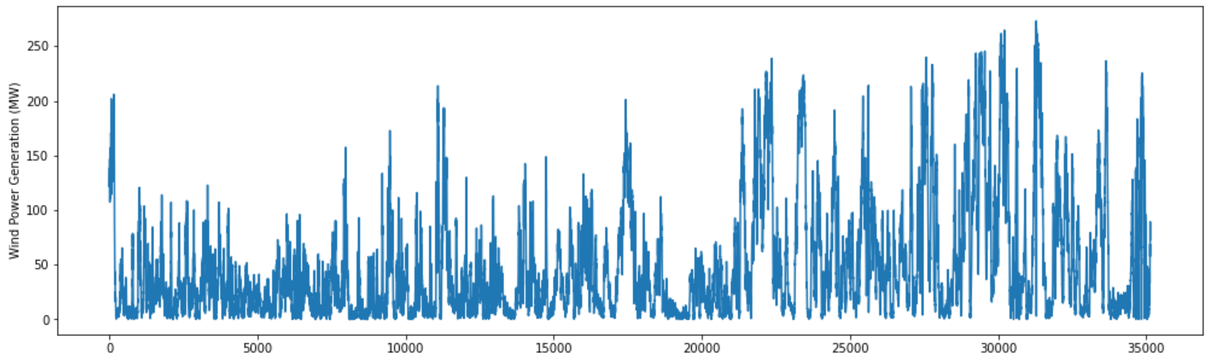


Figure 34. Output variable detail

3.3 Model assumptions

One of the main priorities of this piece of work is that of being able to provide guidelines for choosing the most optimal model from the available ones for each new forecasting problem. Given the vast number of hyperparameters that may be modified within each model, this workstream has been focused on assessing the impact of 4 of them (algorithm, numbers of layers, number of neurons per layer, and leaning rate) and fixing the values of some additional ones based on their overall acceptance according to the consulted bibliography or execution simplicity, given the computing resources used for the development of the models. The following list of hyperparameters has been used equally in all tested models (Grid search, Random search, Bayesian optimization and Genetic Algorithms)

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

The rank of values to be tested in the variable hyperparameters have been randomly chosen and do not correspond to any previous conducted tests. This is done in such way in order to prevent the finals results from being biased and not reflect the real behavior to be expected in the first steps of any prediction problem. Furthermore, in order to try to reproduce the results from one model evaluation to the next one, a random seed has been set.

3.4 Benchmark model

The benchmark model is one of the references that will be used to evaluate final results, together with total execution times (seconds) and sensibility to the models´ hyperparameters initialization (random behavior).

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

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

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

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

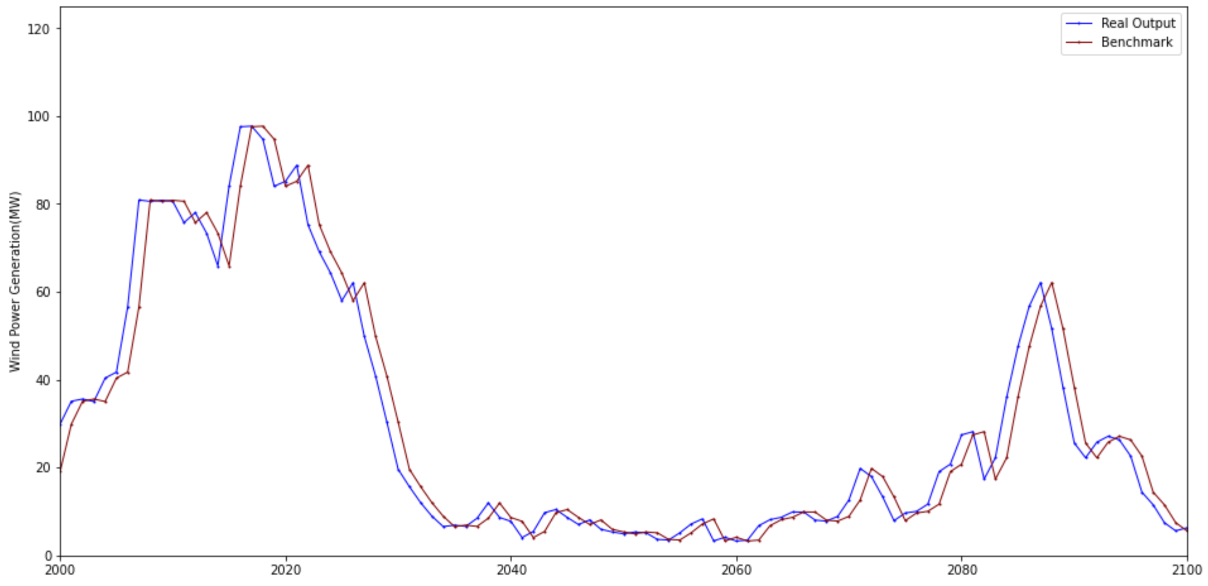


Figure 35. WG -Real output vs Benchmark model

CHAPTER 4. RESULTS

In this chapter, the results obtained from the tested forecasting models are presented and discussed.

4.1 Grid Search tuning technique

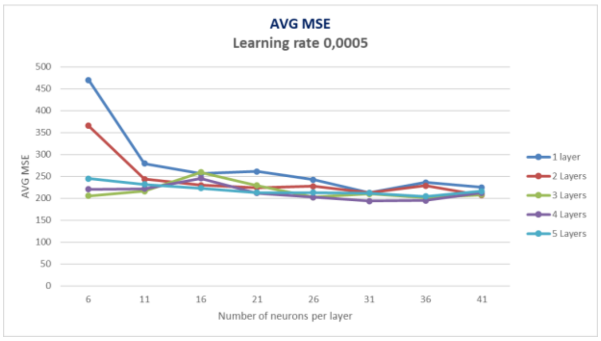
Testing LSTM networks

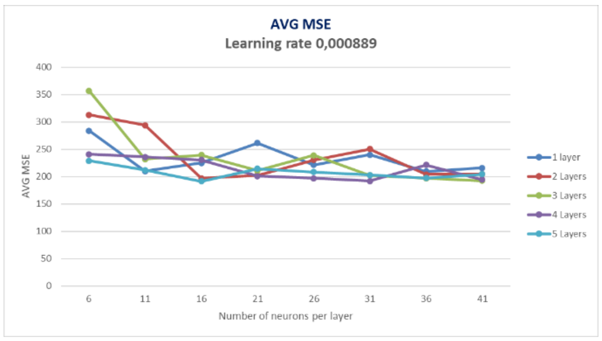
Grid models are the simplest ones from the list of hyperparameter-tuning techniques that this piece of work will test. The first approach, as with the rest of techniques, was to assess its performance using an LSTM algorithm.

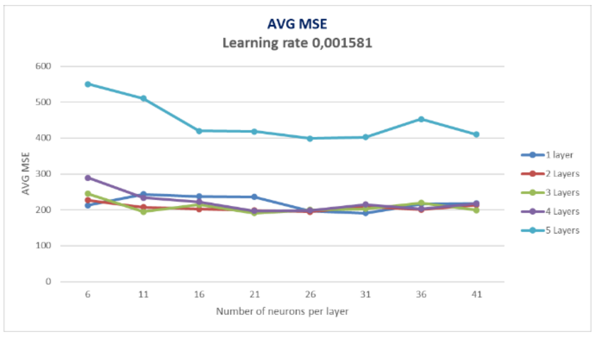
5 different simulations were performed to measure how sensitive the model was to its initialization. Even though this random behavior is usually controlled by setting a random seed to be used equally in all iterations, it was not possible to obtain perfectly reproductible results across all 5 simulations, which adds complexity to the problem by itself. The fact that results vary from one simulation to the next one makes it difficult to recognize when the model might have gotten stuck in a local optima or whether it has achieved the minimum achievable error for the available data.

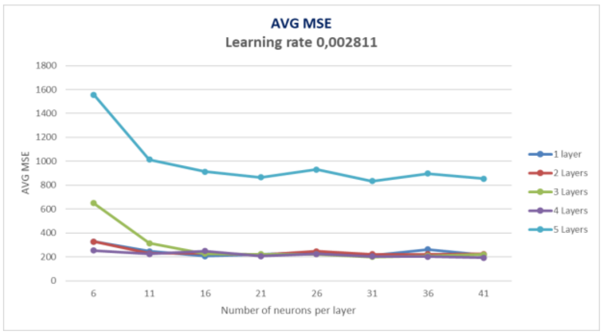
Even though this happens across all tuning techniques, grid search models shed some light over this behavior as the model will, in all cases, try all the listed configurations. Due to this, it is possible to compare results from all 5 simulations, even when the final results do not match to perfection.

The following figures show the evolution of the average MSE, computed as the average from all 5 simulations, depending on the number of layers and number of neurons per layer of the network and grouped by the different tested learning rates.









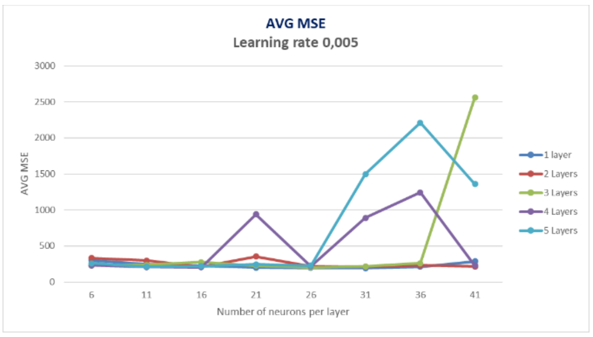
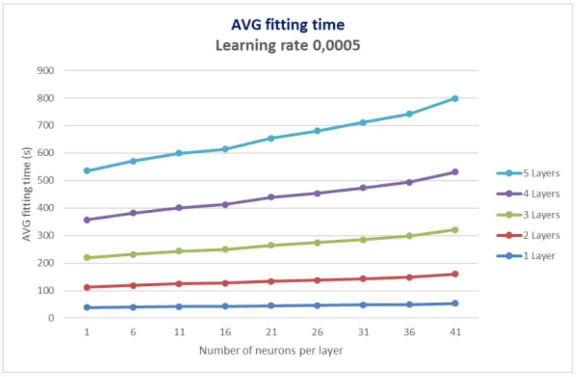


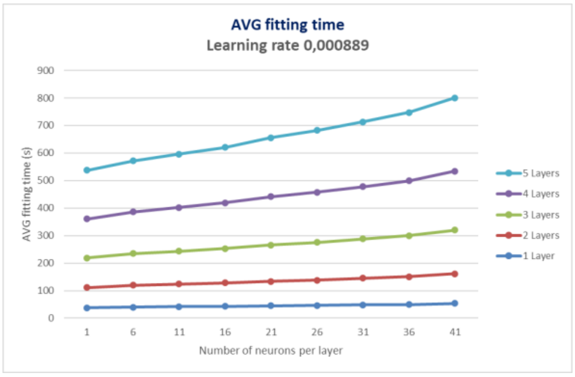
Figure 36. Grid model- LSTM average MSE value

It can be observed there is a certain degree of direct correlation between all 3 hyperparameters and the resulting MSE. As a general behavior, higher learning rate values yield overall higher MSE values, and so this behavior extends to a when a higher number of layers and neurons per layer apply.

These results are consistent since the logic behind these hyperparameters forces this pattern. As previously explained, the learning rate controls the variation of the model weights in each iteration based on the error obtained. If these leaps are increased, there is a risk of "skipping" the optimal minimum and therefore reaching a higher error even using the same hyperparameters. The number of layers and neurons per layer can also have a negative effect when both are increased simultaneously, due to the over-adjustment of the model during training, in what is known as overfitting. Although the model behaves well in the training phase, an overlearning of the temporal function of the data leads to a later increase in error in the validation phase, which is what has been observed in these cases.

But not only is MSE affected by this. Fitting time can also benefit from simpler networks, as depicted in the following figures, in which the same behavior is observed as with regards to MSE, paying special attention to the results obtained with a learning rate of 0,005.





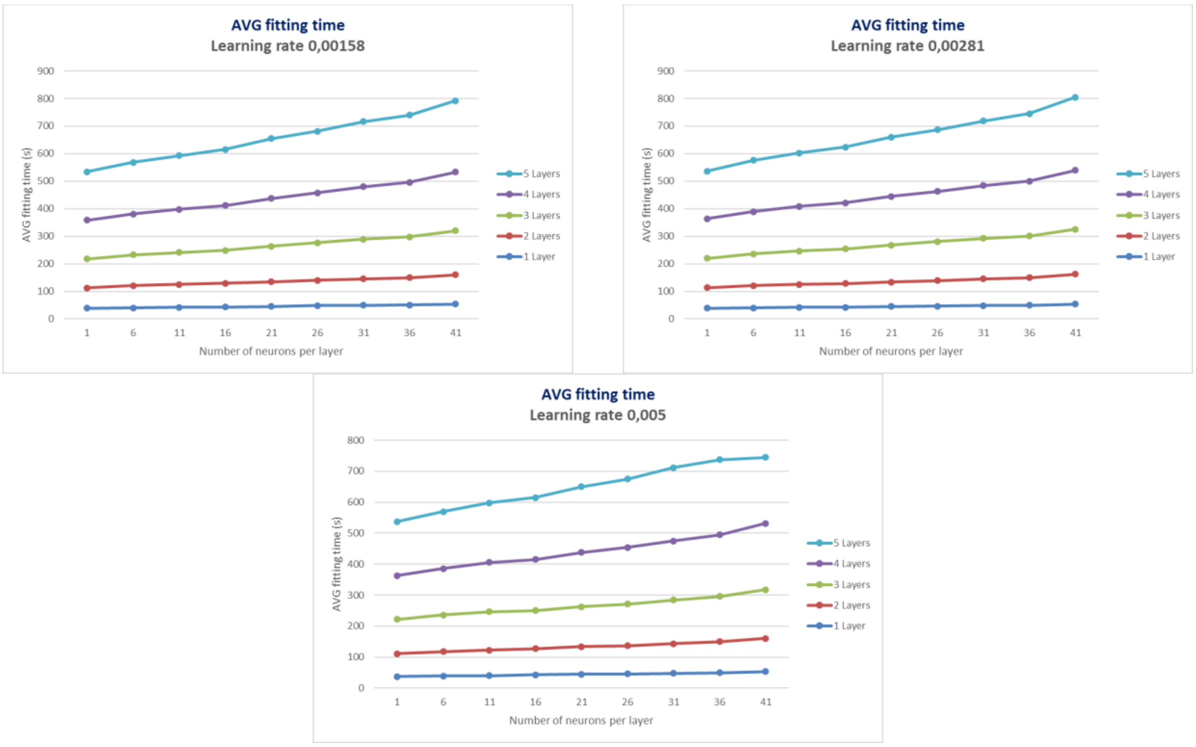


Figure 37. Grid - LSTM average fitting time (s)

On average, all 5 evaluations fitted 225 candidates and took a total 260,21 min to run.

The best performing model was obtained with the following configuration:

Algorithm	Layers	Neurons per layer	Activation	Optimizer	Learning rate	Validation MSE	Training time (s)
lstm	2	26	elu	adam	0,00158114	175,390069	101,375545

Table 1. Best Grid Search-LSTM performing model

Comparing LSTM to GRU, RNN and BiLSTM networks

Although LSTM networks are the most used today, as observed in the research about the state of the art carried out in chapter 2, it is also interesting to see the behavior of the same model under different prediction algorithms.

For this, the Grid Search model was executed for a Gru, RNN and BiLSTM network with the objective of identifying if the use of LSTMs is totally justified over the rest of the algorithms, or if any of the alternatives allows to obtain better results, either in terms of time or accuracy.

Algorithm	Layers	Neurons per layer	Activation	Optimizer	Learning rate	Validation MSE	
lstm	2	26	elu	adam	0,00158114	175,390069	-
gru	3	16	elu	adam	0,0005	167,600379	-4,44%
rnn	3	6	elu	adam	0,0005	179,856083	2,55%
bilstm	2	11	elu	adam	0,005	187,233101	6,75%

Table 2. Accuracy comparison for different network algorithms. Grid Search model.

Algorithm	Layers	Neurons per layer	Activation	Optimizer	Learning rate	Training time (s)		Total execution time (min)	
lstm	2	26	elu	adam	0,00158114	101,375545	-	260,21	-
gru	3	16	elu	adam	0,0005	114,932046	13,37%	217,3	-16,49%
rnn	3	6	elu	adam	0,0005	46,9500928	-53,69%	107,1	-58,84%
bilstm	2	11	elu	adam	0,005	290,645908	186,70%	511,6	96,61%

Table 3. Execution time comparison for different network algorithms. Grid Search model.

Tables 2 and 3 show how GRU networks have an overall better performance over LSTM networks, both in accuracy and total execution time. RNN networks are simpler and faster to execute but have in exchange a slight decrease in accuracy, though it may be overseen in case time was a hard constrain of the model.

4.2 Random Search tuning technique

The following are the results derived from the execution of the random search for LSTM network algorithms.

Contrary to the experienced with the Grid search, the execution of random models is highly sensitive to the initialization of the parameters and, as its name indicates, its execution is based on a stochastic behavior that is difficult to reproduce. That is why one of the biggest challenges when using these models is to be able to identify if the model has become stagnant around a local minimum. The randomness to which these models are subject does not simplify the search for the most optimal architecture either. As seen from various simulations, the optimal network configuration differs greatly from one simulation to another and it is utterly complex to assess how much do two simulations differ as the networks configurations tested in each one of them may not coincide across all 50 possible candidates.

The following table shows the top 5 performing architectures across the 5 performed simulations, allowing to observe the beforementioned random performance.

Simulation	Algorithm	Layers	Neurons per layer	Activation	Optimizer	Learning rate	Validation MSE	Training time (s)
1	lstm	5	41	elu	adam	0,000763	184,5923	264,0119
	lstm	3	6	elu	adam	0,00128	188,7829	122,0483
	lstm	4	16	elu	adam	0,003598	188,8014	167,0225
	lstm	4	36	elu	adam	0,001406	189,8118	200,869
	lstm	2	36	elu	adam	0,001406	189,9404	98,38093
2	lstm	3	26	elu	adam	0,0005757	177,381793	128,509002
	lstm	3	41	elu	adam	0,0006034	179,374294	131,992889
	lstm	5	36	elu	adam	0,00359843	181,87616	246,175346
	lstm	4	21	elu	adam	0,00127977	183,830301	170,114023
	lstm	3	16	elu	adam	0,00140588	184,331674	112,866894
3	lstm	3	31	elu	adam	0,00298181	184,265402	133,618373
	lstm	3	16	elu	adam	0,0018638	186,332919	116,835761
	lstm	4	36	elu	adam	0,00204746	187,699712	192,452965
	lstm	1	26	elu	adam	0,00359843	187,989413	46,0481453

	lstm	3	26	elu	adam	0,005	188,207802	128,392598
4	lstm	4	36	elu	adam	0,0011115	180,838919	191,564501
	lstm	3	11	elu	adam	0,00214597	182,219927	127,661104
	lstm	1	21	elu	adam	0,00101179	182,25622	46,9287081
	lstm	5	11	elu	adam	0,0018638	182,895485	193,387224
	lstm	3	11	elu	adam	0,005	185,966645	114,897116
5	lstm	5	26	elu	adam	0,00092103	179,705409	232,834292
	lstm	3	11	elu	adam	0,00066286	180,991821	123,591654
	lstm	4	11	elu	adam	0,00161873	182,341127	145,697482
	lstm	2	26	elu	adam	0,00134135	185,592467	97,9971871
	lstm	1	31	elu	adam	0,00224922	186,046671	52,3809333

Table 4. Top 5 LSTM performing architectures for Random search models. The green highlighted cell corresponds to the best performing hyperparameter configuration

Out of all of them, simulation 2 stands out as the best performing one and the one showing the highest resemblance to the best architecture to the Grid Search model (Table 1). Even though it could be discussed whether this simulation can be taken as a good performing one, meaning that it has managed to escape any local minimum, its similarity to the best performing Grid Search model can be used to dismiss this idea. Since the grid search is forced to test all configurations, the risk of getting stuck in a local minimum is much lower and therefore a model that converges in grid-like conditions can be assumed to have been able to optimize out of local conjectures.

Algorithm	**Layers**	**Neurons per layer**	**Activation**	**Optimizer**	**Learning rate**	**Validation MSE**	**Training time (s)**
lstm	3	26	elu	adam	0,0005757	177,381793	128,509002

Table 5. Best Random Search-LSTM performing model

Execution time wise, each simulation took on average 45.5 mins to fit all 50 possible candidates, which represents an 82,5% reduction compared to Grid Search model. Even though this seems as a huge reduction, it must be kept in mind that the number of candidates per simulation had to be limited to 50 as this model is much more computationally demanding than Grid Search, where 250 candidates where fitted per simulation. Several trials have been conducted with up to 100 iterations, but Google Collab has proved to be unable to perform them successfully, limiting the tests in this work to a maximum allowance of 50 iterations.

As a rough estimation, even if the execution time remained constant at 0.91 min/candidate, if 250 candidates could have been executed, the total time per simulation would have been around 227.5 minutes, below Grid Search average total execution time, but with the consequent need for greater computing resources and without a clear method for recognizing the best performing samples.

Comparing LSTM to Gru, RNN and BiLSTM networks

The following table contains a comparison between the best performing architectures for Random Search models, using the best performing LSTM architecture results as baseline.

Algorithm	Layers	Neurons per layer	Activation	Optimizer	Learning rate	Validation MSE	
lstm	3	31	elu	adam	0,00298181	184,265402	-
gru	4	26	elu	adam	0,00096535	169,507542	-8,01%
rnn	1	21	elu	adam	0,00195347	191,678969	4,02%
bilstm	1	45	elu	adam	0,00259197	190,741036	3,51%

Table 6. Accuracy comparison for different network algorithms. Random Search model.

Algorithm	Layers	Neurons per layer	Activation	Optimizer	Learning rate	Training time (s)		Total execution time (min)	
lstm	3	31	elu	adam	0,00298181	133,618373		45,5	
gru	4	26	elu	adam	0,00096535	159,869971	19,65%	48,6	6,81%
rnn	1	21	elu	adam	0,00195347	20,7571943	-84,47%	24,7	-45,71%
bilstm	1	45	elu	adam	0,00259197	39,0776045	-70,75%	97,3	113,85%

Table 7.. Execution time comparison for different network algorithms. Random Search model.

As it happened with Grid Search, GRU networks are able to improve model performance, in this case by a greater percentage, although its total execution time proves to be higher than baseline LSTM architecture.

RNN networks show again a tradeoff between accuracy and execution time. Where the first one shows a 4,02% reduction, total execution time is reduced by almost 50% showing that these kinds of networks could be a good alternative in those applications with fast performing requirements and minimal accuracy loss. BiLSTM networks neither does improve accuracy, nor execution time and this might be intrinsically related to its complex architecture which does not allow to properly generalize for this problem statement and requires further computing resources to evaluate all possible candidates. This result has already been observed in the simulations carried out by means of grid search and is opposite to what is expected, at least as far as accuracy is concerned. The ability to look into the future of bilstm networks should help generate a better understanding of the problem and extract the most important variables to ensure improved results, especially working with data from which some periodic behavior can be expected. In order to improve this outcome, it has been tested to enhance the number of epochs, but the computational load involved has not allowed valid results to be derived.

4.3 Bayesian optimization tuning technique

In the same manner as in the preceding sections, the results obtained from the execution of LSTM-type network models together with Bayesian optimization are detailed below.

The selected method to recreate the surrogate model has been the Gaussian method due to its implementational simplicity compared to the TPE or RF, presented in CHAPTER 2.

As with random search, the Bayesian optimization method is highly constrained by the initialization of the weights. The mapping of the hyperparameters that manage to minimize the objective function can be highly influenced by the area initially explored, as well as the consequent tests, risking the possibility of falling into local stagnation.

Likewise, the manner in which the acquisition function is generated is of special importance given that the probabilistic function of hyperparameter testing is intrinsically linked to it, but we do not have at present a metric that allows us to measure how optimal this function is for modeling the problem.

Simulation	Algorithm	Layers	Neurons per layer	Activation	Optimizer	Learning rate	Validation MSE	Training time (s)
1	lstm	3	15	elu	adam	0,00123	187,403	80,9715
	lstm	3	45	elu	adam	0,00216	188,563	98,3572
	lstm	5	30	elu	adam	0,00115	189,628	139,07
	lstm	5	35	elu	adam	0,00415	190,263	146,094
	lstm	4	15	elu	adam	0,0037	190,988	108,445
2	lstm	1	10	elu	adam	0,002793641	185,6270732	24,10919285
	lstm	1	15	elu	adam	0,004857473	189,5508934	25,50009561
	lstm	2	25	elu	adam	0,002610418	189,9400619	49,74845505
	lstm	2	15	elu	adam	0,001872107	190,8041131	47,91214442
	lstm	6	40	elu	adam	0,000661062	193,2397536	166,5127091
3	lstm	3	30	elu	adam	0,000974701	181,8056578	64,82969666
	lstm	5	45	elu	adam	0,000415332	184,8565897	131,4268107
	lstm	6	45	elu	adam	0,000813781	185,8635256	154,1208332
	lstm	6	45	elu	adam	0,000670833	187,1209127	147,5872531
	lstm	5	30	elu	adam	0,000405628	187,5727841	107,7409339
4	lstm	6	10	elu	adam	0,001384426	185,7949288	135,8981235
	lstm	4	15	elu	adam	0,000986948	187,2645519	90,36797476
	lstm	2	15	elu	adam	0,002517593	194,0898802	49,18882966
	lstm	5	20	elu	adam	0,001818079	195,3456652	115,2993312
	lstm	5	30	elu	adam	0,002546102	196,2784461	123,172493
5	lstm	3	15	elu	adam	0,001	182,651	53,0911
	lstm	2	15	elu	adam	0,00117	184,091	36,5889
	lstm	4	15	elu	adam	0,00083	191,413	72,19
	lstm	3	30	elu	adam	0,00234	193,684	57,0136
	lstm	2	15	elu	adam	0,00206	193,833	36,59

Table 8. Top 5 LSTM performing architectures for Bayesian optimization simulations. The green highlighted cell corresponds to the best performing hyperparameter configuration

In this particular case, although simulation 3 is the one that achieves the lowest validation error, the rest of the tests are substantially far from what is expected, based on the results obtained with Grid Search and the best simulation of random search, although the behaviour of this simulation is the most logical one since the iterations, focused on the improvement of the validation error, are performed in a more limited environment that reflects a less stochastic behavior.

Algorithm	Layers	Neurons per layer	Activation	Optimizer	Learning rate	Validation MSE	Training time (s)
lstm	3	30	elu	adam	0,000974701	181,8056578	64,82969666

Table 9. Best Bayesian Optimization-LSTM performing model

The rest of the simulations present greater variability in the tests performed, which suggests that the model could benefit from a greater number of iterations. In this aspect, the Bayesian method improves notably against the random search since it allows to execute 100 iterations per simulation, against the 50 of the random one without experiencing complications due to high complexity, albeit a slight increase in the time required (120 minutes on average). This confirms what was expressed in chapter 2, in which the idea of how a Bayesian method managed to simplify the optimization of hyperparameters by allowing work with a subordinate model was discussed.

Comparing LSTM to Gru, RNN and BiLSTM networks

The following is also a comparison of the best results obtained for the different network architectures tested.

Algorithm	Layers	Neurons per layer	Activation	Optimizer	Learning rate	Validation MSE	
lstm	3	30	elu	adam	0,0009747	181,805658	-
gru	2	30	elu	adam	0,00063344	170,248371	-6,36%
rnn	6	15	elu	adam	0,00073817	185,762487	2,18%
bilstm	1	45	elu	adam	0,00259197	190,741036	4,91%

Table 10.Accuracy comparison for different network algorithms. Bayesian optimization.

Algorithm	Layers	Neurons per layer	Activation	Optimizer	Learning rate	Training time (s)		Total execution time (min)	
lstm	3	30	elu	adam	0,0009747	64,8296967	-	118,01	-
gru	2	30	elu	adam	0,00063344	45,0948706	-30,44%	100,12	-15,16%
rnn	6	15	elu	adam	0,00073817	56,2693636	-13,20%	128,17	8,61%
bilstm	1	45	elu	adam	0,00259197	39,0776045	-39,72%	164,23	39,17%

Table 11. Execution time comparison for different network algorithms. Bayesian optimization

In this scenario, GRU architecture is the only one that manages to surpass the LSTM, achieving better results in both accuracy and total execution time.

On the other hand, RNN architecture results in longer execution times, contrary to the random and grid search, which is surprising, since it could be expected to achieve shorter execution times since it is a simpler network configuration. The bilstm architecture is once again relegated to the last position as it does not achieve any notable improvement . Once again, a modification of the epochs is linked to an exponential increase in the resources required to address the problem, making it difficult to draw conclusions as to why this architecture does not achieve the expected results.

4.4 Genetic algorithms tuning technique

The conditions defined for the execution of the genetic algorithms in this work are detailed below:

- Number of individuals: 20
- Number of generations: 4
- Mutation allowance for number of layers in generation i+1:

$$Number\ of\ layers_{\ gen\ i+1} = Number\ of\ layers_{\ gen\ i} + rand[0,1,2]$$

As for the number of layers, it has been defined that its number from one generation to the next one could only be increased by one or two units, or remain constant, to be chosen randomly

- Mutation allowance for number of neurons per layer in generation i+1:

$$Number\ of\ neurons\ per\ layer_{gen\ i+1} = Number\ of\ neurons\ per\ layer_{gen\ i} + rand[0,9]$$

In order to allow mutation in the number of neurons per layer, a random increase in the next generation of up to 9 units has been defined, without allowing its decrease.

- Mutation allowance for the learning rate:

$$Learning\ rate_{gen\ i+1} = Learning\ rate_{gen\ i} * (0.8 + 0.4 * rand[0,1))$$

Algorithm	Layers	Neurons per layer	Activation	Optimizer	Learning rate	Validation MSE	Training time (s)
lstm	3	15	elu	adam	0,00123	187,403	80,9715
lstm	3	45	elu	adam	0,00216	188,563	98,3572
lstm	5	30	elu	adam	0,00115	189,628	139,07
lstm	5	35	elu	adam	0,00415	190,263	146,094
lstm	4	15	elu	adam	0,0037	190,988	108,445

Table 12. Top 5 performing network configurations. Genetic algorithms

In the case of genetic algorithms, it is really complicated to conclude how positive the results are. If there were already problems in the random search or Bayesian optimization, being a technique that depends on random mutations for the sampling of the optimal hyperparameters, this complexity is magnified and greatly reduces the ability to derive logical conclusions from these results.

Although it seems that the tests have remained in the environment of the expected configurations, it is possible to observe how the learning rate is an order of magnitude higher than that obtained with the previously tested techniques and how, in general, the tests tend towards less optimal configurations.

This can be observed in more detail throughout the tests performed with the genetic algorithms in which a considerable increase in the number of neurons per layer is observed, reaching values up to 86 neurons in GRU type network configuration, which is completely contrary to the configurations obtained through the techniques already tested.

Algorithm	Layers	Neurons per layer	Activation	Optimizer	Learning rate	Validation MSE	
lstm	3	30	elu	adam	0,001460758	187,2479481	-
gru	5	39	elu	adam	0,000723036	175,6013818	-6,22%
rnn	4	86	elu	adam	0,001565222	176,3636213	-5,81%
bilstm	2	69	elu	adam	0,002291607	216,8450307	15,81%

Table 13. Accuracy comparison for different network algorithms. Genetic Algorithms

Algorithm	Layers	Neurons per layer	Activation	Optimizer	Learning rate	Traning time (s)		Total execution time (min)	
lstm	3	30	elu	adam	0,001460758	135,3101137	-	100,37	-
gru	5	39	elu	adam	0,000723036	238,6682661	76,39%	56	-44,21%
rnn	4	86	elu	adam	0,001565222	127,3048062	-5,92%	42	-58,15%
bilstm	2	69	elu	adam	0,002291607	290,6459081	114,80%	142,07	41,55%

Table 14. Execution time comparison for different network algorithms. Genetic Algorithms

4.5 Cross performance comparison

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

Table 15. Cross comparison for the best performing simulations

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

Table 16. Number of adjusted candidates per tuning technique

Table 15 shows a more generic comparison of the best results obtained for each configuration and technique tested.

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

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

Regardless of the type of network architecture employed, it is clear how GRID Search techniques are the most interesting to use in a first approach to the problem. Not only are these techniques the ones that achieve, in a general way, the best results, but they also allow exploring the whole map of hyperparameters defined without being influenced by local minimums. Despite the fact that these models take longer to perform all the iterations, they can be a good first preference for the following reasons:

1. If time is a determinant, they may be used as a means of identifying sub-optimal model simulations that are faster to run, but risk fluctuating around a local minimum
2. They can be used to identify, from a wide initial spectrum, those configurations that will improve the validation error, discarding the less interesting ones.
3. They allow, in a single simulation, to obtain a network configuration that, without being optimal, achieves acceptable error, avoiding local stagnation and that can later be improved, either in a more limited range using grid search again or with any of the other techniques studied
4. It allows us to observe with greater transparency the behaviour of the model when faced with changes in the values of the hyperparameters, unlike other methods whose stochastic nature makes it difficult to understand the results obtained.

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

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

CHAPTER 5. CONCLUSIONS

The objective of this work was to perform an extensive research on deep learning models and its application to time series forecasting. The state of the art showed a significant number of models and techniques available to improve the performance of these models. Hence, a case study was performed to explore these techniques in a real-world dataset. The goal was not to achieve the best possible network model, but to test various hyperparameter optimization techniques in a real example of application in order to derive a series of guidelines that could be useful when facing a new problem of time series prediction.

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

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

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

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

- Regardless of the optimization technique employed, GRU architectures far exceed the expectations placed on LSTMs that were initially positioned as those from which to expect the best results, while biLSTMs are relegated to last position by failing to achieve good results in any of the fields tested.
- As far as possible, start the search for hyperparameters in limited environments and opt for intermediate values, avoiding any extreme values. As can be seen from the final table of results (Table 15), most of the results oscillate around 2-3 layers, a number of neurons per layer not exceeding 30 (with some exceptions) and with downward values as far as learning rate is concerned.

  The number of layers and neurons per layer should be adapted to each application, being aware that for more complex applications a greater number of layers and neurons would be required to model the output behavior. It is recommended to perform a more comprehensive initial search, and limit subsequent trials in a limited rank around those values within the optimal configuration appears to be located.
- In order to choose the best hyperparameter optimization method, the following approach can be followed:
  - Is time a decisive factor?

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

- Is the best possible accuracy being sought?

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

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

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

CHAPTER 6. REFERENCES

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

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

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

[4] T. life Editorial, ‘How artificial neural networks copy the brain so AI can think faster than you’, *Medium*, Feb. 14, 2017. https://toa.life/how-artificial-neural-networks-copy-the-brain-and-power-ai-to-think-faster-than-you-218929fa5dd3 (accessed Jan. 19, 2020).

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

[6] ‘Deep Learning Fundamentals - Cognitive Class’. https://cognitiveclass.ai/courses/introduction-deep-learning (accessed Feb. 08, 2020).

[7] J. Brownlee, ‘How to Choose Loss Functions When Training Deep Learning Neural Networks’, *Machine Learning Mastery*, Jan. 29, 2019. https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/ (accessed Feb. 08, 2020).

[8] G. Drakos, ‘How to select the Right Evaluation Metric for Machine Learning Models: Part 1 Regression Metrics’, *Medium*, Feb. 05, 2020. https://medium.com/@george.drakos62/how-to-select-the-right-evaluation-metric-for-machine-learning-models-part-1-regrression-metrics-3606e25beae0 (accessed Feb. 08, 2020).

[9] P. Grover, ‘5 Regression Loss Functions All Machine Learners Should Know’, *Medium*, Feb. 05, 2020. https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0 (accessed Feb. 09, 2020).

[10] M.-A. Maiza, ‘The Unknown Benefits of using a Soft-F1 Loss in Classification Systems’, *Medium*, Dec. 05, 2019. https://towardsdatascience.com/the-unknown-benefits-of-using-a-soft-f1-loss-in-classification-systems-753902c0105d (accessed Feb. 09, 2020).

[11] ‘Losses - Keras Documentation’. https://keras.io/losses/ (accessed Feb. 09, 2020).

[12] J. Brownlee, ‘Loss and Loss Functions for Training Deep Learning Neural Networks’, *Machine Learning Mastery*, Jan. 27, 2019. https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/ (accessed Feb. 09, 2020).

[13] ‘François Chollet, J.J. Allaire - Deep Learning with R-Manning Publications (2017).pdf’. .

[14] R. Akita, A. Yoshihara, T. Matsubara, and K. Uehara, ‘Deep learning for stock prediction using numerical and textual information’, in *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*, Okayama, Japan, Jun. 2016, pp. 1–6, doi: 10.1109/ICIS.2016.7550882.

[15] Y. Wu *et al.*, ‘Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation’, *arXiv:1609.08144 [cs]*, Oct. 2016, Accessed: Jan. 06, 2020. [Online]. Available: http://arxiv.org/abs/1609.08144.

[16] D. M. Q. Nelson, A. C. M. Pereira, and R. A. de Oliveira, ‘Stock market’s price movement prediction with LSTM neural networks’, in *2017 International Joint Conference on Neural Networks (IJCNN)*, Anchorage, AK, USA, May 2017, pp. 1419–1426, doi: 10.1109/IJCNN.2017.7966019.

[17] K. Chen, Y. Zhou, and F. Dai, ‘A LSTM-based method for stock returns prediction: A case study of China stock market’, in *2015 IEEE International Conference on Big Data (Big Data)*, Santa Clara, CA, USA, Oct. 2015, pp. 2823–2824, doi: 10.1109/BigData.2015.7364089.

[18] S. Bouktif, A. Fiaz, A. Ouni, and M. Serhani, ‘Optimal Deep Learning LSTM Model for Electric Load Forecasting using Feature Selection and Genetic Algorithm: Comparison with Machine Learning Approaches †’, *Energies*, vol. 11, no. 7, p. 1636, Jun. 2018, doi: 10.3390/en11071636.

[19] S. Chen and H. He, ‘Stock Prediction Using Convolutional Neural Network’, *IOP Conf. Ser.: Mater. Sci. Eng.*, vol. 435, p. 012026, Nov. 2018, doi: 10.1088/1757-899X/435/1/012026.

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

[21] Xiaosheng Peng *et al.*, ‘A very short term wind power prediction approach based on Multilayer Restricted Boltzmann Machine’, in *2016 IEEE PES Asia-Pacific Power and Energy Engineering Conference (APPEEC)*, Xi’an, China, Oct. 2016, pp. 2409–2413, doi: 10.1109/APPEEC.2016.7779917.

[22] L.-C. Cheng, Y.-H. Huang, and M.-E. Wu, ‘Applied attention-based LSTM neural networks in stock prediction’, in *2018 IEEE International Conference on Big Data (Big Data)*, Seattle, WA, USA, Dec. 2018, pp. 4716–4718, doi: 10.1109/BigData.2018.8622541.

[23] S. A. Ludwig, ‘Comparison of Time Series Approaches applied to Greenhouse Gas Analysis: ANFIS, RNN, and LSTM’, in *2019 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, New Orleans, LA, USA, Jun. 2019, pp. 1–6, doi: 10.1109/FUZZ-IEEE.2019.8859013.

[24] M. Dorraki, A. Fouladzadeh, A. Allison, B. Coventry, and D. Abbott, ‘Deep Learning for C-Reactive Protein Prediction’, in *2018 2nd European Conference on Electrical Engineering and Computer Science (EECS)*, Bern, Switzerland, Dec. 2018, pp. 160–164, doi: 10.1109/EECS.2018.00037.

[25] R. Kaabi, M. Sayadi, M. Bouchouicha, F. Fnaiech, E. Moreau, and J. M. Ginoux, ‘Early smoke detection of forest wildfire video using deep belief network’, in *2018 4th International Conference on Advanced Technologies for Signal and Image Processing (ATSIP)*, Sousse, Mar. 2018, pp. 1–6, doi: 10.1109/ATSIP.2018.8364446.

[26] Jian Zheng, Cencen Xu, Ziang Zhang, and Xiaohua Li, ‘Electric load forecasting in smart grids using Long-Short-Term-Memory based Recurrent Neural Network’, in *2017 51st Annual Conference on Information Sciences and Systems (CISS)*, Baltimore, MD, USA, Mar. 2017, pp. 1–6, doi: 10.1109/CISS.2017.7926112.

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

[28] A. Kulaglic and B. Berk Üstüngag, ‘Stock price forecast using Wavelet Transformations in Multiple time Windows and Neural Networks’. IEEE, 2018.

[29] J. Brownlee, ‘An Introduction to Feature Selection’, *Machine Learning Mastery*, Oct. 05, 2014. https://machinelearningmastery.com/an-introduction-to-feature-selection/ (accessed Jan. 20, 2020).
[30] R. Agarwal, ‘The 5 Feature Selection Algorithms every Data Scientist should know’, *Medium*, Jul. 28, 2019. https://towardsdatascience.com/the-5-feature-selection-algorithms-every-data-scientist-need-to-know-3a6b566efd2 (accessed Jan. 26, 2020).
[31] J. Brownlee, ‘How to Choose a Feature Selection Method For Machine Learning’, *Machine Learning Mastery*, Nov. 26, 2019. https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/ (accessed Jan. 26, 2020).
[32] xiaoharper, ‘Filter Based Feature Selection - ML Studio (classic) - Azure’. https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/filter-based-feature-selection (accessed Jan. 26, 2020).
[33] ‘Pearson Product-Moment Correlation - When you should run this test, the range of values the coefficient can take and how to measure strength of association.’ https://statistics.laerd.com/statistical-guides/pearson-correlation-coefficient-statistical-guide.php (accessed Jan. 26, 2020).
[34] ‘Spearman’s Rank-Order Correlation - A guide to when to use it, what it does and what the assumptions are.’ https://statistics.laerd.com/statistical-guides/spearmans-rank-order-correlation-statistical-guide.php (accessed Jan. 26, 2020).
[35] J. Magiya, ‘Kendall Rank Correlation Explained.’, *Medium*, Nov. 23, 2019. https://towardsdatascience.com/kendall-rank-correlation-explained-dee01d99c535 (accessed Jan. 26, 2020).
[36] sampath kumar gajawada, ‘ANOVA for Feature Selection in Machine Learning’, *Medium*, Oct. 20, 2019. https://towardsdatascience.com/anova-for-feature-selection-in-machine-learning-d9305e228476 (accessed Feb. 06, 2020).
[37] sampath kumar gajawada, ‘Chi-Square Test for Feature Selection in Machine learning’, *Medium*, Oct. 20, 2019. https://towardsdatascience.com/chi-square-test-for-feature-selection-in-machine-learning-206b1f0b8223 (accessed Jan. 26, 2020).
[38] J. Brownlee, ‘Information Gain and Mutual Information for Machine Learning’, *Machine Learning Mastery*, Oct. 15, 2019. https://machinelearningmastery.com/information-gain-and-mutual-information/ (accessed Jan. 27, 2020).
[39] D. Soni, ‘Introduction to Evolutionary Algorithms’, *Medium*, Jul. 16, 2019. https://towardsdatascience.com/introduction-to-evolutionary-algorithms-a8594b484ac (accessed Jan. 21, 2020).
[40] P. Casas, ‘Feature Selection using Genetic Algorithms in R’, *Medium*, Apr. 08, 2019. https://towardsdatascience.com/feature-selection-using-genetic-algorithms-in-r-3d9252f1aa66 (accessed Jan. 21, 2020).
[41] V. Mallawaarachchi, ‘Introduction to Genetic Algorithms — Including Example Code’, *Medium*, Nov. 20, 2019. https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3 (accessed Jan. 21, 2020).
[42] S. Patnaik, Ed., *Recent Developments in Intelligent Nature-Inspired Computing:* IGI Global, 2017.
[43] X. Z. Gao, V. Govindasamy, H. Xu, X. Wang, and K. Zenger, ‘Harmony Search Method: Theory and Applications’, *Computational Intelligence and Neuroscience*, 2015. https://www.hindawi.com/journals/cin/2015/258491/ (accessed Jan. 27, 2020).
[44] M. Kuhn, *20 Recursive Feature Elimination | The caret Package*. .
[45] Q. Chen, Z. Meng, X. Liu, Q. Jin, and R. Su, ‘Decision Variants for the Automatic Determination of Optimal Feature Subset in RF-RFE’, *Genes*, vol. 9, no. 6, p. 301, Jun. 2018, doi: 10.3390/genes9060301.

[46] 'sklearn.feature_selection.RFECV — scikit-learn 0.22.1 documentation'. https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFECV.html#sklearn.feature_selection.RFECV (accessed Jan. 28, 2020).
[47] 'Sequential Feature Selector - mlxtend'. http://rasbt.github.io/mlxtend/user_guide/feature_selection/SequentialFeatureSelector/ (accessed Jan. 28, 2020).
[48] DataVedas, 'EMBEDDED METHODS | Data Vedas'. https://www.datavedas.com/embedded-methods/ (accessed Jan. 28, 2020).
[49] T. Lin, T. Guo, and K. Aberer, 'Hybrid Neural Networks for Learning the Trend in Time Series', in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, Melbourne, Australia, Aug. 2017, pp. 2273–2279, doi: 10.24963/ijcai.2017/316.
[50] '(1) (PDF) An Adaptive Offloading Method for an IoT-Cloud Converged Virtual Machine System Using a Hybrid Deep Neural Network', *ResearchGate*. https://www.researchgate.net/publication/328636008_An_Adaptive_Offloading_Method_for_an_IoT-Cloud_Converged_Virtual_Machine_System_Using_a_Hybrid_Deep_Neural_Network (accessed Mar. 01, 2020).
[51] 'images (Imagen PNG, 270 × 187 píxeles)'. https://encrypted-tbn0.gstatic.com/images?q=tbn%3AANd9GcS6o-bFFynY9S_urprOHPLGPf_MWzRWR8dTohZujIhl9gdRKuKm&usqp=CAU (accessed Apr. 23, 2020).
[52] 'A Beginner's Guide to Attention Mechanisms and Memory Networks', *Pathmind*. http://pathmind.com/wiki/attention-mechanism-memory-network (accessed Apr. 23, 2020).
[53] 'Hyperparameters: Optimization Methods and Real World Model Management', *MissingLink.ai*. https://missinglink.ai/guides/neural-network-concepts/hyperparameters-optimization-methods-and-real-world-model-management/ (accessed Feb. 03, 2020).
[54] J. Brownlee, 'Understand the Impact of Learning Rate on Neural Network Performance', *Machine Learning Mastery*, Jan. 24, 2019. https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/ (accessed Feb. 04, 2020).
[55] 'machine learning - Epoch vs Iteration when training neural networks', *Stack Overflow*. https://stackoverflow.com/questions/4752626/epoch-vs-iteration-when-training-neural-networks (accessed Feb. 04, 2020).
[56] 'Hyperparameter optimization for Neural Networks — NeuPy'. http://neupy.com/2016/12/17/hyperparameter_optimization_for_neural_networks.html#hand-tuning (accessed Feb. 05, 2020).
[57] A. Bissuel, 'Hyper-parameter optimization algorithms: a short review', *Medium*, Apr. 24, 2019. https://medium.com/criteo-labs/hyper-parameter-optimization-algorithms-2fe447525903 (accessed Feb. 05, 2020).
[58] 'The Unreasonable Effectiveness of Quasirandom Sequences | Extreme Learning'. http://extremelearning.com.au/unreasonable-effectiveness-of-quasirandom-sequences/ (accessed Feb. 05, 2020).
[59] W. Koehrsen, 'A Conceptual Explanation of Bayesian Hyperparameter Optimization for Machine Learning', *Medium*, Jul. 02, 2018. https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization-for-machine-learning-b8172278050f (accessed Feb. 05, 2020).
[60] 'HyLAP - Meta-Data'. http://www.hylap.org/meta_data/adaboost/ (accessed Feb. 05, 2020).

[61] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, ‘Algorithms for Hyper-Parameter Optimization’, p. 9.

[62] A. Osipenko, ‘Genetic algorithms and hyperparameters — Weekend of a Data Scientist’, *Medium*, May 30, 2019. https://medium.com/cindicator/genetic-algorithms-and-hyperparameters-weekend-of-a-data-scientist-8f069669015e (accessed Feb. 06, 2020).

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

[64] dpicampaigns, ‘About the Sustainable Development Goals’, *United Nations Sustainable Development*. https://www.un.org/sustainabledevelopment/sustainable-development-goals/ (accessed Aug. 22, 2020).

CHAPTER 7. ANNEXES

ANNEX I - SUSTAINABLE DEVELOPMENT GOALS

The sustainable development goals, defined in 2015 by the UN and approved by 193 countries, are 17 interconnected goals that encompass a total of 169 targets valid until 2030, which recognize the need to address both the fight against poverty, care for the planet and the reduction of inequalities [64].

These objectives require the collaboration of not only civil society, but also the public and private sectors to achieve them, thus making the world more diverse and egalitarian.

This work mainly supports the following sustainable development objectives and their corresponding targets [64]:

7 Affordable and clean energy

7.1 By 2030, ensure universal access to affordable, reliable and modern energy services
7.2 By 2030, increase substantially the share of renewable energy in the global energy mix
7.3 By 2030, double the global rate of improvement in energy efficiency
7.4 By 2030, enhance international cooperation to facilitate access to clean energy research and technology, including renewable energy, energy efficiency and advanced and cleaner fossil-fuel technology, and promote investment in energy infrastructure and clean energy technology
7.5 By 2030, expand infrastructure and upgrade technology for supplying modern and sustainable energy services for all in developing countries, in particular least developed countries, small island developing States, and land-locked developing countries, in accordance with their respective programs of support

9 Industries innovation and infrastructure

9.1 Develop quality, reliable, sustainable and resilient infrastructure, including regional and transborder infrastructure, to support economic development and human well-being, with a focus on affordable and equitable access for all
9.2 Promote inclusive and sustainable industrialization and, by 2030, significantly raise industry's share of employment and gross domestic product, in line with national circumstances, and double its share in least developed countries
9.3 Increase the access of small-scale industrial and other enterprises, in particular in developing countries, to financial services, including affordable credit, and their integration into value chains and markets
9.4 By 2030, upgrade infrastructure and retrofit industries to make them sustainable, with increased resource-use efficiency and greater adoption of clean and environmentally sound technologies and industrial processes, with all countries taking action in accordance with their respective capabilities
9.5 Enhance scientific research, upgrade the technological capabilities of industrial sectors in all countries, in particular developing countries, including, by 2030, encouraging innovation and substantially increasing the number of research and development workers per 1 million people and public and private research and development spending
9.6 Facilitate sustainable and resilient infrastructure development in developing countries through enhanced financial, technological and technical support to African countries,

least developed countries, landlocked developing countries and small island developing States 18

9.7 Support domestic technology development, research and innovation in developing countries, including by ensuring a conducive policy environment for, inter alia, industrial diversification and value addition to commodities

9.8 Significantly increase access to information and communications technology and strive to provide universal and affordable access to the Internet in least developed countries by 2020

13 Climate action

13.1 Strengthen resilience and adaptive capacity to climate-related hazards and natural disasters in all countries

13.2 Integrate climate change measures into national policies, strategies and planning

13.3 Improve education, awareness-raising and human and institutional capacity on climate change mitigation, adaptation, impact reduction and early warning

13.4 Implement the commitment undertaken by developed-country parties to the United Nations Framework Convention on Climate Change to a goal of mobilizing jointly $100 billion annually by 2020 from all sources to address the needs of developing countries in the context of meaningful mitigation actions and transparency on implementation and fully operationalize the Green Climate Fund through its capitalization as soon as possible

13.5 Promote mechanisms for raising capacity for effective climate change-related planning and management in least developed countries and small island developing States, including focusing on women, youth and local and marginalized communities

This work supports these objectives by providing a research framework to enhance and improve the robustness of forecasting techniques, the application of which can assist in the adoption of renewable generation methods as unique sources of electricity.

The use of reliable forecasting techniques for the short to medium term is essential both for entering the electricity markets and for undertaking maintenance work in plants or power stations.

The market factor is highly relevant, since the variability in power generation from renewable plants is much greater than that of traditional competitors. This may imply penalties in the event of not being able to generate the offered energy. The fact that the generation capacity of each park can be predicted with a greater degree of precision means that the various renewable generation technologies can be regarded as strong competitors in the electricity market.

Operation and maintenance of the farms is an equally important economic aspect. Achieving optimal maintenance of the infrastructure is key to ensuring a quality service, fast and reliable and with a broad useful life, ensuring a high degree of amortization, leading to a call effect for future investments.