FEDERAL UNIVERSITY OF RIO GRANDE DO SUL
INSTITUTE OF INFORMATICS
POSTGRADUATE PROGRAM IN COMPUTING

RAFAEL THOMAZI GONZALEZ

Using Deep Learning and Evolutionary Algorithms for Time Series Forecasting

Final Paper presented in partial fulfillment of the requirement for the degree of Master of Computer Science.

Supervisor: Prof. Dr. Dante Augusto Couto Barone

Porto Alegre
2018

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Gonzalez, Rafael Thomazi
Using Deep Learning and Evolutionary Algorithms for Time Series Forecasting / Rafael Thomazi Gonzalez. – 2018.
15 f.:il.
Orientador: Dr. Dante Augusto Couto Barone.
Dissertação (Mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2018.
1.Deep Learning. 2.Evolutionary Algorithm. 3.Time series forecasting. I. Barone, Dante Augusto Couto. II. Using Deep Learning and Evolutionary Algorithms for Time Series Forecasting.

ABSTRACT

Time series analysis is widely used in fields such as business, economics, finance, science, and engineering. One of the main purposes of time series data analysis is to use past observations from the data to forecast future values. Moreover, time series data analysis allows you to represent the data in a form that can convey changes over time. Many different time series forecasting algorithms have been explored in machine learning and statistics literature. More recently, deep neural networks have been increasingly used, since they can be trained in such a way that they are effective at representing many kinds of data, including raw and featurized data.

This thesis aims to assess the performance of Deep Learning algorithms optimized by an Evolutionary Algorithm in predicting different time series. First, a description of the selected Deep Learning algorithms will be presented, namely Stacked Autoencoder (SAE), Stacked Denoising Autoencoder (SDAE) and Long Short-Term Memory Networks (LSTM). The Feedforward Multilayer Perceptron (MLP) network is used frequently in time series prediction, and thus it is used as baseline to compare these Deep Learning models. Given the complexity of these models, their hyperparameters are optimized by an Evolutionary Algorithm called Covariance Matrix Adaptation Evolution Strategy (CMAES). The strengths and drawbacks of CMAES are also highlighted in order to explain why it is considered as state-of-the-art and one of the most powerful Evolutionary Algorithms for real-valued optimization.

In order to demonstrate the performance of the proposed approach on forecasting time series, experiments are performed using three different datasets. Two of them are artificial data generated by the Mackey-Glass and Lorenz System equations. The third one includes real data of hourly energy demand. Throughout the analysis of the results, it was found that some models, such as LSTM and MLP, perform better on data presenting some degree of seasonality; while models with pre-processing layers (i.e. SAE and SDAE) have difficulties learning the time structure of the data.

Problems containing time series data behave similar to many other machine learning problems such that there is no master algorithm that is the best for all problems. Therefore, this study supports the effectiveness of deep learning models for usage on time series forecasting problems, as well as the usage of CMAES for hyperparameters optimization.

Keywords: Deep Learning. Evolutionary Algorithm. Time series forecasting.

Usando Aprendizagem Profunda e Algoritmos Evolutivos para Previsão de Séries Temporais

RESUMO

A análise de séries temporais é amplamente utilizada em areas relacionadas a negócios, economia, finanças, ciências e engenharia. Uma das principais caracteristicas dos dados de séries temporais é que observações passadas podem ser usadas para prever valores futuros. Além disso, esse tipo de dado introduze o problema adicional de se fazer necessário a criação de representações que reflitam mudanças ao longo do tempo. Muitos algoritmos de previsão de séries temporais baseados em aprendizado de máquina e estatística têm sido propostos na literatura. Mais recentemente, tecnincas de Deep Learning vêm sendo aplicadas nesse campo, uma vez que esses tipos de rede neurais podem ser treinadas de forma a representarem diferentes tipos de dados, sejam dados brutos ou transformados.

Esta tese tem por objetivo avaliar o desempenho de algoritmos de Aprendizagem Profunda otimizados por um Algoritmo Evolutivo na predição de diferentes séries temporais. Primeiramente, é apresentada uma descrição dos algoritmos de Aprendizado Profundo selecionados, a saber: Autoencoder (SAE), Stacked Denoising Autoencoder (SDAE) e redes Long Short-Term Memory (LSTM). A rede Feedforward Multilayer Perceptron (MLP) é usada freqüentemente em predições de séries temporais e, portanto, é usada como modelo base para comparar os modelos base em Aprendizagem Profunda. Dada a complexidade desses modelos, seus hiperparâmetros são otimizados por um Algoritmo Evolucionário denominado Covariance Matrix Adaptation Evolution Strategy (CMAES). Os pontos fortes e as desvantagens do CMAES são destacados a fim de se explicar por que ele é considerado como estado-da-arte e um dos mais poderosos algoritmos evolutivos para otimização de valor real.

Para demonstrar o desempenho da abordagem proposta na previsão de séries temporais, os experimentos são realizados usando três conjuntos de dados diferentes. Dois deles são dados artificiais gerados pelas equações de Mackey-Glass e Lorenz System. O terceiro inclui dados reais de demanda de energia horária. Ao longo da análise dos resultados, verificou-se que alguns modelos, como o LSTM e o MLP, apresentam melhor desempenho em dados que apresentam algum grau de sazonalidade; enquanto os modelos com camadas de pré-processamento (ou seja, SAE e SDAE) têm dificuldades em aprender a estrutura temporal dos dados.

Os problemas que envolvem dados de séries temporais se comportam de maneira semelhante a muitos outros problemas de aprendizado de máquina, de modo que não há um

algoritmo que seja o melhor para todos os problemas. Portanto, este trabalho corrobora a eficácia da utilização de modelos de Aprendizagem Profunda em problemas de previsão de séries temporais, bem como a eficácia do uso do algoritmo CMAES na otimização de hiperparâmetros.

Palavras-chave: Aprendizagem Profunda. Algoritmo Evolutivo. Previsão de séries temporais.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABREVIATIONS

ACF	Autocorrelation Function
ADF	Augmented Dickey-Fuller
AE	Autoencoder
ANN	Artificial Neural Network
ARMA	Autoregressive Moving Average models
BPTT	Backpropagation Through Time
CMAES	Covariance Matrix Adaptation Evolution Strategy
CNN	Convolutional Neural Network
CSA	Cumulative Step-size Adaptation
CV	Cross-Validation
DAE	Denoising Autoencoder
DBN	Deep Belief Network
DL	Deep Learning
DNN	Deep Neural Network
EA	Evolutionary Algorithm
GA	Genetic Algorithm
LSTM	Long Short-Term Memory
MAPE	Mean Absolute Percentage Error
MLP	Multilayer Perceptron
MSE	Mean Squared Error
PSO	Particle Swarm Optimization
RMSE	Root Mean Squared Error
RNN	Recurrent Neural Network
SAE	Stacked Autoencoder
SDAE	Stacked Denoising Autoencoder
SVM	Support Vector Machine
SVR	Support Vector Regressor
TSF	Time Series Forecasting
UL	Unsupervised Learning

SUMARY

1 INTRODUCTION

This chapter presents the needed background information to understand the remainder of this work. It is introduced the time series forecasting and hyperparameter optimization problems. It also includes the problem statement, objectives and contributions of this work.

1.1 Time Series Forecasting

A time series is an ordered sequence of data points, usually measured in uniform time intervals (NIST/SEMATECH, 2013). An important property of time series is that data observations are interdependent and, thus, it is essential to maintain the order in which the data were generated (Wei, 2006). There are a number of characteristics of time-series data that make it different from other types of data. Firstly, the sampled data often contain much noise and have high dimensionality, which make them challenging to analyze and model. To deal with this, signal processing techniques such as dimensionality reduction and filtering can be applied to remove some of the noise and reduce the dimensionality. However, valuable information could be lost and the choice of features and signal processing techniques may require expertise of the data. Secondly, it is not certain that there is enough information available to understand the process. For example, in financial data when observing a single stock, which only measures a small aspect of a complex system, there is most likely not enough information in order to predict the future (Fama, 1965). Further, time-series have an explicit dependency on the time variable. Thus, another challenge is that the length of the time-dependencies could be unknown. To solve this problem, the model either has to include more data from the past or must have a memory of past inputs. Many time-series are also non-stationary, meaning that the characteristics of the data, such as mean, variance, and frequency, changes over time. It is very troubling to deal with time series with a high degree of non-stationary, thus transforming the original data in order to make it more stationary, in general, improve the forecasting performance.

Time series analysis comprises methods or processes that breakdown a series into components and explainable portions that allows trends to be identified, estimates and forecasts to be made. The use of observations from a time series available at time t to predict its value at time $t + l$ is called forecasting; where l is called the forecasting horizon. The forecasting horizon is the number of time steps in the future for which the forecasts must be produced. Time

Series Forecasting (TSF) attempts to understand the underlying context of the data points through the use of a model to forecast future values based on known past values. Learning fast dynamics and canceling noise simultaneously is a challenge related directly to the tradeoff between underfitting and overfitting. Indeed, learning noise causes potentially overfitting, whereas, forgetting fast dynamics leads to underfitting. Time series prediction is a very important practical problem with several applications from economic and business planning to signal processing and control. Time series are found in many important areas including communication, health, finance, and natural sciences. Large-scale computer and communication networks generate time-varying metric that characterize the state of the networks. Audio signal arriving at a microphone or radio-frequency signal arriving at a receive antenna are series of values as well. End-of-day face-values of different financial instruments are time-series. In all these cases, it is of interest to understand and model patterns, understand how they impacts, or they are impacted, by other factors, and forecast future values (NIST/SEMATECH, 2013).

Analysis of time-series data has been the subject of active research for decades (Dietterich, 2002; Keogh and Kasetty, 2002) and is considered as one of the top 10 challenging problems in data mining due to its unique properties (Yang and Wu, 2006). A number of techniques have been developed in an attempt to predict time series. Starting from a simple linear Autoregressive Moving Average models (ARMA) (Lütkepohl, 2005) up to the complex non-linear models (Casdagli, 1989), the idea is similar: establish the regression-based description of the future series based on the historical data series. More recently a number of Machine Learning techniques started to be applied to time series forecasting and on a number of occasions showed considerable improvement compared to traditional regression models (Vojinovic et al., 2001; Tay and Cao, 2002). Artificial Neural Networks (ANNs) are particularly good at capturing complex non-linear characteristics of time series (Lee Giles et al., 2001). Support Vector Machine represents another powerful regression technique that immediately found applications in time series forecasting (Mukherjee et al., 1997; Gonzalez et al., 2015).

However, more complex, high-dimensional, and noisy real-world time-series data cannot be described with analytical equations with parameters to solve since the dynamics are either too complex or unknown. Traditional shallow methods, which contain only a small number of non-linear operations, do not have the capacity to accurately model such complex data (Taylor, 2010). Thus, in order to better model complex real-world data, one approach is to develop robust features that capture the relevant information. However, developing domain-specific features for each task is expensive, time-consuming, and requires expertise of the data.

There is an increasing interest in learning the representation from unlabeled data instead of using hand-designed features. Unsupervised feature learning has shown to be successful at learning layers of feature representations for static datasets. This technique helped Deep Neural Networks (DNNs) to be trained more easily, starting a new subarea of study in artificial intelligence often called Deep Learning (DL). Deep learning refers broadly to models that derive meaning out of data by using a hierarchy of multiple layers that mimic the neural networks of human brain. Each layer progressively extracts higher-level (more abstract) features that could be more relevant for the final classification/regression task (Bengio, 2013). Deep Learning models have been obtaining state-of-art results in various problems, both in academy and in industry, i.e. in speech recognition and signal processing (Seide et al., 2011), pattern recognition (Cireşan et al., 2010), and natural language processing (Collobert et al., 2011). Even with some works available, which are reviewed in Chapter 2, the use of Deep Learning models in time series forecasting has received less attention.

1.2 Hyperparameter Optimization

A common trait in Machine Learning models is that they are parameterized by a set of hyperparameters, which are used to configure various aspects of the algorithm and can have varying effects on the resulting model and its performance. For example, considering an ANN, one can optimize the number of layers, the number of units in each layer, the learning rate, and so on. Finding the optimal set of hyperparameters is one of the major challenges when using machine learning methods, since these parameters are directed related to the complexity and the performance of the model (i.e. the more complex the model, the greater is the risk of overfitting).

Hyperparameter selection is typically approached as a non-differentiable, single-objective optimization problem over a mixed-type, constrained domain. The objective is to find a value that minimizes a loss function $L(T, M)$ for a model M on a training set T, sometimes under certain constraints. This model M is constructed by a learning algorithm A using T, and typically involves solving an optimization problem. The model may be parameterized by the hyperparameters, and it is defined as $M = A(T, \lambda)$. The goal of the hyperparameter selection is to find the parametrization λ^* that yields a desired model M^*, while minimizing $L(V, M^*)$, where V is the validation set. Formally, it becomes (Claesen and De Moor, 2015):

$$\lambda^* = argmin_\lambda L(T, M) = argmin_\lambda f(\lambda, A, T, V, L) \quad (1)$$

The objective function f takes the hyperparameters, and returns the associated loss value. The datasets T and V (where $T \cap V = \emptyset$) are given, and the learning algorithm A along with the loss function L are chosen beforehand. The generalization ability of the trained model is quantified using the test set (unseen during the optimization). Then, this problem can be turned into a trade-off between the exploration (traversing the unknown regions of the space, where the classification performance on V is unknown), and exploitation (analyzing the hyperparameters which will likely perform well, and are close to the already-investigated "good" hyperparameters).

The best set of hyperparameters is commonly defined manually by an "educated guess" or by applying a grid or random search over predefined search space (Bergstra et al., 2011). Since the hyperparameter space is in general large and evaluating the objective function (i.e. the performance of the model with a given set of hyperparameters) is computationally expensive, the need for a more efficient way of searching this space arises. Sophisticated methods for hyperparameter optimization have been proposed in the literature, such as Bayesian methods that are considered one of the most effective methods (Eggensperger et al., 2013; Snoek et al., 2012; Williams and Rasmussen, 2006). However, the cost of evaluating the objective function of this optimization problem can be very expensive, which often makes using sequential hyperparameters optimization techniques unfeasible. In order to overcome this issue, Evolutionary Algorithms (EAs), which are presented in Chapter 5, can be used in this type of derivative-free continuous optimization problem since it allows for perfect parallelization of the evaluation of the objective function (Loshchilov and Hutter, 2016). A deeper review of applications of these methods for hyperparameter optimization is presented in Section 2.2. In specific, the algorithm called Covariance Matrix Adaptation Evolution Strategy (CMAES), which is considered as state-of-the-art and one of the most powerful evolutionary algorithms for real-valued optimization (Hansen and Ostermeier, 2001), is presented in Section 5.1. This algorithm is used in this work for optimizing the hyperparameters of the selected deep learning models.

1.3 Objective and Contributions

Although there has been extensive research carried out in the time series forecasting field, the search for simple and fast algorithms that produce reliable forecast has never ceased. The current state-of-the art algorithms in the forecasting domain still have limitations. For example, traditional statistical models such as the autoregressive AR (p) and the moving average MA (q) are linear models. They are appropriate for only some type of time series data (e.g. autoregressive models are more appropriate for stationary series) and are unable to extract complex relationships from the data (Box et al., 2015).

In the context of time series forecasting, this work aims to provide a review and a comparison of Deep Learning algorithms when applied on this kind of problems. According to related works the same method implementations can perform differently for different data sets. Therefore, a comparative study of the performance of four algorithms is conducted using synthetic and real data. The synthetic series are generated using the Mackey-Glass and Lorenz System functions. The real data represents the hourly energy demand of the New England zone, US, provided by ISO New England. Considering that very few related works compare different Deep Learning algorithms on different datasets, the broader goals of this project is to further improve the actual literature on time series forecasting and generate more insights on how these models perform on time series data with different characteristics.

In this work, we tackle the problem of the automated hyperparameter selection in Deep Neural Networks using a Covariance Matrix Adaptation Evolution Strategy (CMAES) algorithm. CMAES has been shown to be extremely effective in solving multiple tasks in many fields, and it has a big potential for the large-scale parallelization. Finally, combinatorial and real-valued optimization problems are very well suited for CMAES. Therefore, we propose this approach for navigating through the large hyperparameter spaces and retrieving the desired parametrization of the DNNs.

1.4 Outline of the Thesis

This work is divided in eight chapters, including the introduction and conclusion. The remainder of this volume is organized as follows: Chapter 2 reviews some related works in Deep Learning and Evolutionary Algorithms. It first presents state-of-the-art approaches that apply Deep Learning models to forecast time series data, and then it is outlined applications of Evolutionary Algorithms in the task of hyperparameter optimization; Chapter 3 introduces basic concepts of Artificial Neural Network and Recurrent Neural Networks, which are two

architectures often used to build deep networks; Chapter 4 delves into the principles of Deep Learning, outlining the concepts of the models used in this work; Chapter 5 presents the foundations of the Evolutionary Algorithms, along with the key concepts of the CMAES algorithm that is applied to optimize the hyperparameters of the forecasting models studied in this work; In Chapter 6, the experimental procedure is explained, as it presents the benchmark datasets, data modeling process, performance evaluation criteria, and tools used to perform the experiments; Chapter 7 discuss the experimental results, and lastly the conclusions and suggestions for future projects are drawn in Chapter 8.

2 RELATED WORKS

This study builds upon previous studies in the area of time series forecasting, deep learning, and hyperparameter optimization techniques. In the following, we focus the review of previous work on Deep Learning and Evolutionary Algorithms applied to time series forecasting, as the proposal of this work is based on the integration of such techniques. In this chapter, from research literature, applications of time series forecasting models is first presented in Section 2.1. The standard statistical forecasting models and the machine learning forecasting models, especially deep neural networks, are presented. Finally, in Section 2.2, some applications of evolutionary algorithms in hyperparameter selection are reviewed.

2.1 Time Series Forecasting

Time Series analysis has been a popular subject of interest in many fields such as economics, engineering and medicine. Traditional techniques on manipulating such data can be found in (Hamilton, 1994). Statistical methods have a much longer history than the machine learning methods. Nowadays many researchers questioned the efficiency of statistical models for solving some real-life problems in comparison with machine learning methods. Artificial Neural Networks have proved to perform well on time series forecasting problems, being widely studied at literature (Azoff, 1994; Chakraborty et al., 1992). Deep Learning models, that were originally used in classification and pattern recognition problems, started to be applied in various machine learning tasks, among which, time series forecasting. Given that this work builds upon the application of Deep Learning models to time series forecasting problems, this section proceeds to a review of previous studies in this area.

Romeu et al. (Romeu et al., 2013) used Deep Neural Networks to predict indoor temperatures series. It was used Stacked Denoising Autoencoders (SDAE) as unsupervised pre-training layers. The experiments considered the fine tuning that occurred only in the last layer or in all network layers. DNNs with pre-training steps outperformed systems without pre-training, but not as much as in other types of problems. This could be justified by the characteristic of the series used and the low-dimensional data.

Similarly, in (Lv et al., 2015) it is proposed a traffic flow prediction system based on deep architecture models with big traffic data. In this work, it is used Stacked Autoencoders (SAE) to learn generic traffic flow features considering the spatial and temporal correlations

inherently. The final model is trained in a greedy layer-wise fashion. The experiments demonstrate that the proposed method for traffic flow prediction achieved superior performance when compared to other shallow models, which indicates that it is more suitable for real-world applications.

Kuremoto et al. (Kuremoto et al., 2014) used a Deep Belief Network (DBN), which is a probabilistic generative neural network composed by multiple layers of Restricted Boltzmann Machine (RBM), to predict different competition time series. The proposed model consists of a 3-layer deep network of RBMs. According to the results, the performance of DBN was better than the one of models such as MLP (Multilayer Perceptron), Bayesian learning and ARIMA. The performance of the DBN was improved with the use of transformed data. Some hyperparameters of the model, such as lag of the time series, number of neurons in the hidden layer, and learning rates was optimized using the algorithm called Particle Swarm Optimization (PSO) during the training process.

Weather forecasting is a popular application of time series forecasting. In (Grover et al., 2015) the authors tackle this problem by also using DBN. The proposed architecture is hybrid model that combines a bottom-up prediction of a set of weather-related variables (wind speed, temperature, pressure, and dew point across space and time) with a top-down Deep Belief Network that models the joint statistical relationships of these variables. The experiments on real-world meteorological data shows that the proposed methodology can provide better results than known benchmarks, as well as recent research that had demonstrated improvements over these benchmarks.

The recurrent neural networks have been used before for forecasting Household Electricity Consumption (Marvuglia and Messineo, 2012). The research was motivated by the concern about increasing electricity consumption. The electricity demand of Sicily was analyzed from 2001 to 2010. For prediction, Elman neural networks were implemented with the same number of context units as hidden units. For this data several networks were tested and the best network was chosen according to prediction accuracy. The model was implemented in order to forecast the electric current intensity at time t. It is one hour ahead forecasting. Additionally, to the historical series of the hourly mean values of electric current intensity, exogenous inputs have been also added. They have added weather variables (temperature, relative humidity...), a variable that takes in account approximate number of air-conditioners available in that area, and a variable that reflects discomfort rate according to temperature and relative humidity. The stopping criteria of the training was the lowest error achieved at the validation data. To find the best architecture, a different number on hidden nodes, various

combination of input values, and different number of lag values were used. Also, before training all input values were normalized in the range [-1,1]. The best network's mean percentage prediction error computed for a test week was 1.5%. The authors also suggested that performance accuracy might be improved with more accurate input data. Also adding exogenous values made a big difference.

Regarding LSTM network, networks using LSTM cells have offered better performance than standard recurrent unit. Thierou et al. (Thireou and Reczko, 2007) applied a bidirectional LSTM to the sequence-based prediction of subcellular proteins localization. The algorithm outperforms feedforward and standard recurrent networks solving the same problem. Moreover, these networks have given state-of-the-art results in different areas, such as phoneme recognition (Graves et al., 2013), optical character recognition (OCR) (Breuel et al., 2013), language identification (Zazo et al., 2016), and text-to-speech synthesis (Fan et al., 2014).

As reviewed above, literature has some works on Deep Learning with time series data. However, there are still no conclusive results about the role of pre-training on this model and the relationship between architecture and the dimensionality of the data. In the case of time series data, the dimensionality is usually provided by the lags of the time series. Moreover, deep networks have been used to achieve state-of-the-art results on a number of benchmark data sets and for solving difficult AI tasks. However, much focus in the feature learning community has been on developing models for static data and not so much on time-series data.

2.2 Evolutionary Algorithms and hyperparameter optimization

As previously pointed out in Section 1.2, hyperparameter selection is a very important step in building machine learning models. DNNs learned using backpropagation have achieved a remarkable level of success recently, even outperforming humans at many different tasks. However, these models are very dependent on their parametrization and often require experts to determine which hyperparameters to modify and how should they be tuned, meaning that for a non-expert it might be hard to find good settings. The need of designing automatic methods for determining the hyperparameters is especially important for increasingly complex deep neural networks architectures for which trial-and-error is unfeasible. Evolutionary Algorithms have been shown very efficient in solving different challenging optimization problems (Nalepa and Kawulok, 2014). Therefore, this section aims to reviewing proposals that address the problem of hyperparameter optimization using this kind of optimization technique.

In my previous work (Gonzalez et al., 2015), it is used a Genetic Algorithm (GA) to perform both hyperparameters optimization and feature selection for an ensemble of Support Vector Machines. The experimental results demonstrated that using a GA improved the performance and stability of the ensemble of SVM. Furthermore, the proposed method outperformed other well-known ensemble methods in a time series classification task. Tsai et al. (Tsai et al., 2006) also used a GA to tune both network structure and parameters of a feedforward neural network, i.e. he numbers of hidden nodes and the links of each node. The performance of the proposed GA-based network is evaluated using different problems that have numerous local optima. The experiments show that the presented GA approach can obtain better results than the existing method reported recently in the literature.

Another Evolutionary Algorithm called Particle Swarm Optimization (PSO) (Kennedy and Eberhart, 1995) has also been applied to hyperparameters optimization problems. Kuremoto et al. (Kuremoto et al., 2014) used this algorithm to optimize the hyperparameters of a DBN, such as lag of the time series, the number of neurons in the hidden layer, and learning rates. The proposed method outperforms other ANNs both in known benchmark and chaotic datasets. Lorenzo et al. (Lorenzo et al., 2017) also demonstrated that PSO can efficiently explore the solution space for this task. This work showed that DNNs of a minimal topology optimized by PSO can obtain competitive classification performance over different datasets.

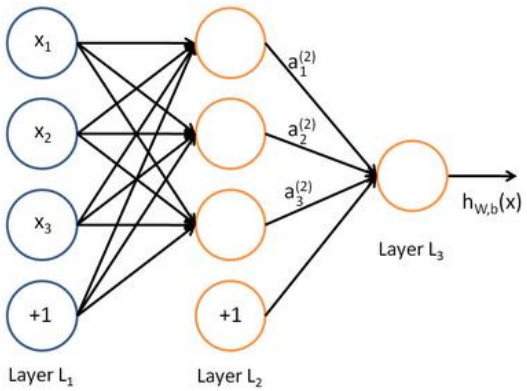
As an alternative, Loshchilov et al. (Loshchilov and Hutter, 2016) propose to use the Covariance Matrix Adaptation Evolution Strategy, which is known for its state-of-the-art performance in derivative-free optimization. In that work, CMAES is used to optimize the hyperparameters of a Convolutional Neural Network (CNN). The proposed study compared the performance of the CMAES against various Bayesian optimization methods. The results presented that CMAES obtains state-of-the-art results on a image classification task, outperforming other optimization methods.

3 ARTIFICIAL NEURAL NETWORKS

An Artificial Neural Network (ANN) is a computational model biologically inspired in the information-processing structures of the human brain (Jain et al., 1996). ANNs processes information through a series of interconnected computational nodes called neurons or perceptrons. These computational nodes are grouped into layers and are associated with one another using weighted connections which represent the "memory" of the system. The structure of ANNs itself allows the transformation of the input space. The consecutive layers perform a cascade of nonlinear transformations that distort the input space allowing the data to become more easily separable.

A Feedforward Neural Network, or Multilayer Perceptron (MLP), is an ANN with a fully-connected (or dense) topology, in which each neuron in one layer is connected with every neuron of the previous layer (Haykin, 1999). Figure 3.1 presents a three-layer MLP. The first layer (L_1) is the input layer which has the same number of neurons as the size of the input vector. The middle layer of nodes (L_2) is called the hidden layer, because its values are not observed in the training set. The third layer (L_3) is the output layer which aggregates the outputs from the hidden layer neurons and outputs a hypothesis ($h_{w,b}(x)$). Moreover, the output value (activation) of neuron i in layer l is denoted by $a_i^{(l)}$. For $l = 1$, we use $a_i^{(1)} = x_1$ to denote the $i - th$ input of the network.

Figure 3.1 – A three-layer Neural Network



Source: Ng et al. 2013b (Ng et al., 2013b)

Traditionally, neural networks can learn through a gradient descent-based algorithm. The gradient descent algorithm aims to find the values of the network weights that best minimize the error (difference) between the estimated and true outputs. Since the MLP

architecture can have several layers, in order to adjust all the weights along the hidden layers, it is necessary to propagate this error backward (from the output to the input layer). This propagation procedure is called *Backpropagation* (Rumelhart et al., 1986), and allows the network to estimate how much the weights from the lower layers need to be changed by the gradient descent algorithm. Initially, when a neural network is trained, the weights are set at random. When the training set is presented to the network, the data is propagated through the nonlinear transformation along the layers. The estimated output is then compared to the true output, and the error is propagated from the output towards the input, allowing the gradient descent algorithm to adjust the weights as required. The process continues iteratively until the error has reached its minimum value.

ANNs have been successfully applied to different problems, such as, pattern classification (the task of assigning an input pattern to one of many prespecified classes), function approximation (finding an estimated value of an unknown function), and forecasting (given a set of labeled training patterns in a time sequence, predict the value of a sample at future time) (Jain et al., 1996).

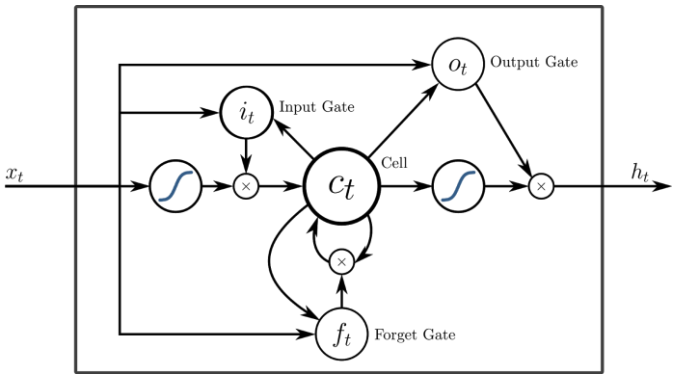
3.1 Recurrent Neural Networks

A limitation of the MLP architecture is that it assumes that all inputs and outputs are independent of each other. In order for an MLP to model a time series, it is necessary to include some temporal information in the input data. Recurrent Neural Networks (RNNs) are neural networks specifically designed to tackle this problem, making use of a recurrent connection in every unit. The activation of a neuron is fed back to itself with a weight and a unit time delay, which provides it with a memory of past activations, which allows it to learn the temporal dynamics of sequential data (Hüsken and Stagge, 2003). They are called recurrent because they perform the same task for every element of a sequence; thus, the output of the network depends on the previous computations. The short-term time-dependency is modelled by the hidden-to-hidden connections without using any time delay-taps. It is possible to adapt the Backpropagation algorithm to train a RNN by “unfolding” the network through time and constraining some of the connections to always hold the same weights, this procedure is known as Backpropagation-through-time (BPTT) (Pascanu et al., 2013).

While in principle the recurrent network is a simple and powerful model, in practice, it is unfortunately hard to train properly. The main problem that arises from the unfolding of a

RNN is that the gradient of some of the weights starts to become too small or too large if the network is unfolded for too many time steps. These are called the vanishing or exploding gradients problems (Bengio et al., 1994). A type of network architecture that solves this problem is the Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997). LSTMs extend RNN with memory cells, instead of recurrent units, to store and output information, easing the learning of temporal relationships on long time scales. LSTMs make use of the concept of gating: a mechanism based on component-wise multiplication of the input, which defines the behavior of each individual memory cell. These gates trap the error in the cell and keep the gradient steep enough during the training process, forming a so-called "error carrousel". A LSTM memory cell is composed of three gates: an input gate (i_t), a forget gate (f_t), and an output (o_t) gate. The input gate controls the impact of the input value on the state of the memory cell. The output gate controls the impact of the state of the memory cell on the output at the current time step. The forget gate determines how much of prior memory value should be passed into the next time step. Depending on the states of these gates, LSTM can represent long-term or short-term dependency of sequential data. Figure 3.2 shows an illustration of a LSTM cell.

Figure 3.2 – LSTM cell with input (i_t), forget (f_t), and output (o_t) gates

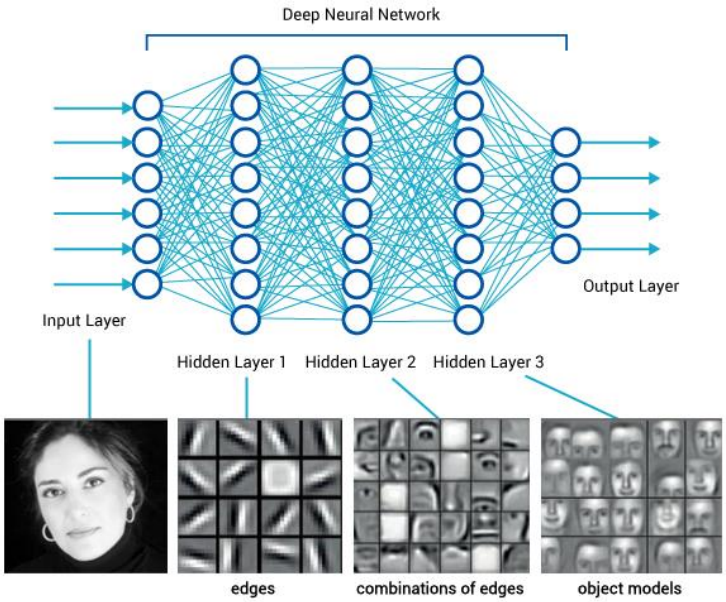


Source: Greff et al., 2015 (Greff et al., 2015)

4 DEEP LEARNING

Artificial intelligence has developed significantly in recent years as machine learning and classification algorithms have become more sophisticated and powerful. Currently, one of the most popular and promising approaches to machine learning is Deep Learning (DL) (Bengio, 2013). Deep learning involves learning the hierarchical structure of data by initially learning simple low-level features which are in turn used to successively build up more complex representations, capturing the underlying regularities of the data. In contrast to previous machine learning algorithms, Deep Learning algorithms are constituted by multiple hierarchical hidden layers between the input and output layers. Those hidden layers are composed of units that can be used to describe underlying features of the data. Benefits of hierarchical neural networks persist in a drastically improved ability to recognize patterns resulting in more reliable applications. In a common facial recognition task, as presented in Figure 4.1, the input layer represents the pixels of the image while the output is the corresponding identity (or classification) of the face, while the hidden layers can represent low-level features, such as edges and shapes, to high-level features, such as “big eyes” or “short hair”.

Figure 4.1 – Deep Neural Network applied on a facial recognition task



Source: Yali, 2018 (Yali, 2018)

Learning the structure of a deep architecture aims to automatically discover these abstractions, from the lowest to highest levels. Favorable learning algorithms would be

unsupervised, depending on minimal human effort, while allowing the network to discover these latent variables on its own, rather than requiring a predefined set of all possible abstractions. The ability to achieve this task while requiring little human input is particularly important for higher-level abstractions as humans are often unable to explicitly identify potential hidden, underlying factors of the raw input (Bengio, 2009). Thus, the power to automatically learn important underlying features made deep architectures so popular.

4.1 Unsupervised Learning

Deep Learning techniques became practically feasible to some extent through the help of Unsupervised Learning (UL). In this context, Unsupervised Learning studies how systems can learn to represent particular input patterns in a way that reflects the statistical structure of the overall collection of input patterns. These methods work only with the observed input data, thus there are no explicit target outputs or environmental evaluations associated with each input; rather the unsupervised learner brings to bear prior biases as to what aspects of the structure of the input should be captured in the output (Barlow, 1989).

At the beginning, training deep networks using Backpropagation did not show good results. The problem stemmed from the fact that as a layer eventually learned a task reasonably well, the learned features were not successfully propagated to successive layers in the network. In these models, the information of the error becomes increasingly smaller as it propagates backward from the output to the input layer, to a point where initial layers do not get useful feedback on how to adjust their weights. This issue was called “the vanishing gradient problem”. In 1992, Hochreiter’s mentor, Jürgen Schmidhuber, attempted to solve this problem by organizing a multi-level deep hierarchy which could be effectively pre-trained one level at a time via random initialization and unsupervised learning, followed by a supervised Backpropagation pass for fine-tuning (Schmidhuber, 1992). This method allows each level of the hierarchy to learn a compressed representation of the input observation which is in turn fed into the next level as the successive input. However, while deep architectures were promising, the issue remained that many poor results were suggesting that gradient-based training of randomly initialized supervised deep neural networks easily got stuck in local minima or plateaus as the architecture got deeper (Bengio et al., 2006). In 2006, however, Hinton and colleagues revolutionized the DL field by presenting the idea of “greedy layer-wise training” algorithm for pre-training each layer of a deep network using an unsupervised approach (Hinton

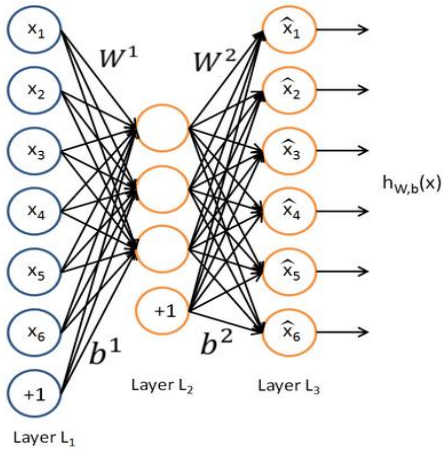
et al., 2006). This method consists of two steps: 1) an unsupervised step, where each layer is trained individually and 2) a supervised step, where the previously trained layers are stacked, one additional output layer is added, and the whole network parameters are fine-tuned using the backpropagation algorithm. This breakthrough led to the fast-growing interest in Deep Learning and enabled the development of models that yielded state-of-the-art results in tasks such as handwritten digits classification.

4.1.1 Autoencoder

An Autoencoder (AE) is an Unsupervised Learning algorithm that is trained to minimize the discrepancy between the input data and its reconstruction. In other words, it is trying to learn an approximation to the identity function, so as to output $\hat{x}$ that is similar to x (Ng et al., 2013a). It does this by setting the target values to be equal to the inputs and applying the backpropagation algorithm on the network. Intuitively, if a representation allows a good reconstruction of its input, it means that it has retained much of the information that was present in that input.

Autoencoders are comprised of two main components. The first component, i.e. the "encoder", learns to generate a latent representation of the input data, whereas the second component, i.e. the "decoder", learns to use these learned latent representations to reconstruct the input data as close as possible to the original. In its shallow structure, as shown in Figure 4.2, an autoencoder is comprised of three layers. The first layer (L_1) represents the original input data. The second layer (L_2) is the encoded representation of the input data. Finally, the third layer (L_3) represents the reconstruction of the input. Moreover, it has a set of parameters (W, b), where (W^1, b^1) represents the weights and biases of the encoder network, and (W^2, b^2) represents the weights and biases of the decoder network.

Figure 4.2 – A single layer autoencoder

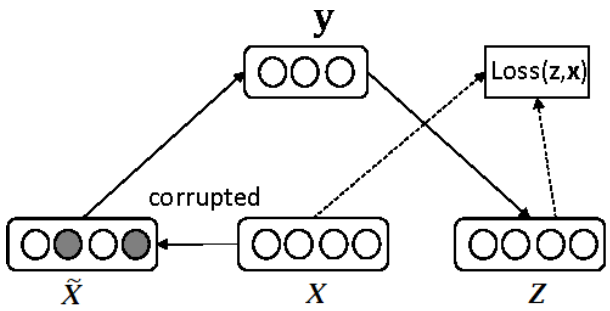


Source: Ng et al., 2013a (Ng et al., 2013a)

Learning the identity function seems a particularly trivial task, but placing constraints on the network can reveal interesting structure about the data. An example of a constraint is a limitation to the number of hidden units in the hidden layer, thus forcing the network to learn a compressed representation of the input. This method allows for the discovery of internal representations of the data that rely on fewer intermediate features. For example, for a facial recognition task, each pixel of the image may be represented at the input layer. That data is compressed in the hidden layer into features such as “small mouth” or “wide eyes.” That is, the input data of the face can be described using less data than is actually given in the image. That compressed data can then be uncompressed in order to re-represent the input data at the output layer, allowing the facial image to be reconstructed entirely from the learned features. However, even when the number of hidden layers is large (even larger than the number of input values), it is still possible to discover interesting structures on the data, by applying a sparsity constraint on the hidden layer (Ng et al., 2013a). A sparse autoencoder has very few hidden neurons that are active. A neuron in an artificial neural network is considered “active” if its output value is close to 1, while it is considered “inactive” if its output value is close to 0. The concept of creating a sparse autoencoder involves constraining the neurons to be inactive most of the time. As a result, even with many hidden units, the data is constrained, forcing the network to learn the important features of the data in order to reconstruct it. Besides, using dense compressed representations tend to entangle information (i.e., changing a single aspect of the input yields significant changes in all components of the representation), thus sparse representations can be expected to be easier to interpret and to use for a subsequent classifier (Vincent et al., 2010).

Rather than constrain the representation, another possibility is to change the training reconstruction criterion for learning to extract useful features: cleaning partially corrupted input, or in short denoising. In doing so the implicit definition of a good representation is modified into the following: “a good representation is one that can be obtained robustly from a corrupted input and that will be useful for recovering the corresponding clean input”. It is expected that a higher-level representation should be rather stable and robust under corruptions of the input. This approach leads to a variant of the basic autoencoder described above: the Denoising Autoencoder (DAE) (Vincent et al., 2010). This version is trained to reconstruct a clean “repaired” input from a corrupted version of it. This is done by first corrupting the initial input X into $\tilde{X}$. Then, the corrupted input $\tilde{X}$ is mapped, as with the basic autoencoder, to a hidden representation y from which the output Z is reconstructed. Figure 4.3 shows the basic structure of a Denoising Autoencoder. It is important to note, that this version is still trained to minimize the same reconstruction loss between a clean X and its reconstruction Z. The difference is that it forces the learning of a far cleverer mapping than the identity: one that extracts features useful for denoising.

Figure 4.3 – The structure of a Denoising Autoencoder



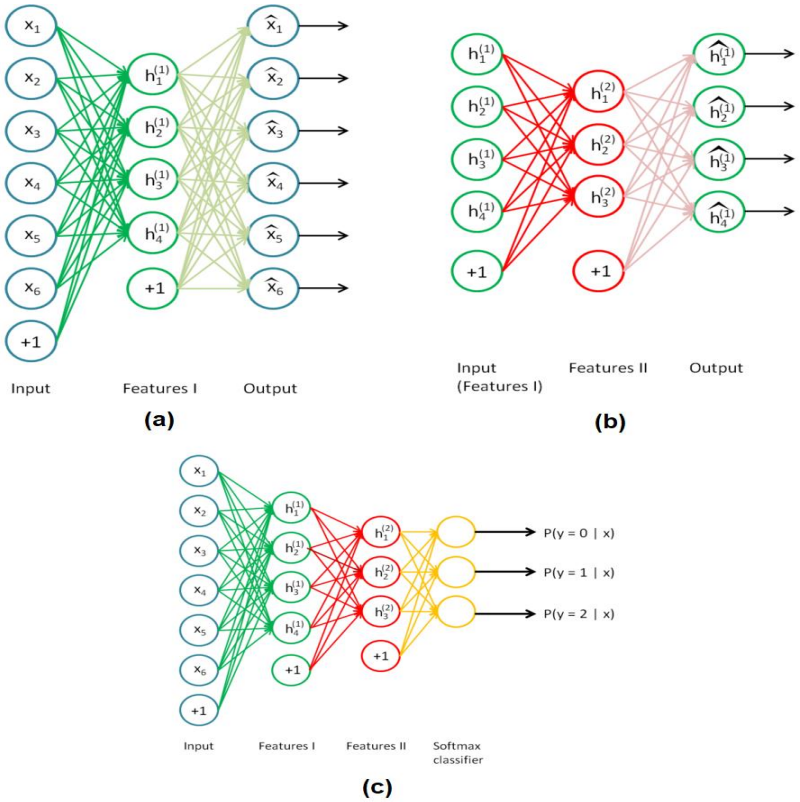
Source: Gonzalez (2018)

Two corruption techniques, which are considered the simplest and most popular ones, are studied in this work: Additive Gaussian noise and Masking noise (Vincent et al., 2010). Additive Gaussian noise is a very common noise mode for real valued inputs. This technique adds a random value that has a Gaussian distribution with 0 mean to the input data. The Masking noise is a common technique for handling missing values. It sets to 0 a fraction of elements of X chosen at random, which can be viewed as turning off components considered missing or replacing their value by a default value. Since all information about these masked components is thus removed from that particular input pattern, it can be said that a DAE that uses this technique is trained to fill-in these artificially introduced “blanks”.

4.2 Stacked Autoencoder

Based on the fact that Autoencoders are automatic features extractors, they can also be stacked to create a deep structure to increase the level of abstraction of learned features. Thus, a Stacked Autoencoder (SAE) is a neural network consisting of multiple layers of Autoencoders (Bengio, 2009). In this case, the network is pre-trained, i.e. each layer is treated as a shallow Autoencoder, generating latent representations of the input data. These latent representations are then used as input for the subsequent layers before the full network is fine-tuned using standard supervised learning algorithm to minimize the error in predicting the supervised target (e.g., class) (Bengio et al., 2006). Figure 4.4 exemplify this training procedure. First, as shown in Figure 4.4a, an Autoencoder is trained on the raw input x_i to learn primary features $h_i^{(1)}$. After this, as shown in Figure 4.4b, these primary features are used as input to a second Autoencoder to learn secondary features $h_i^{(2)}$. Finally, as presented in Figure 4.4c, all three layers are combined together and an output layer is added on top of the stack. With this deep architecture, learning-feature hierarchies are formed by using lower-level learned features to compose higher levels of the hierarchy. The first layer of a SAE tends to learn first-order features in the raw input (such as edges in an image), the second layer tends to learn second-order features corresponding to patterns in the appearance of first-order features (for example, contour or corner detectors) and, following this logic, higher layers to learn even higher-order features (Bengio et al., 2006).

Figure 4.4 – Stacked Autoencoder training process



Source: Ng et al. 2013c (Ng et al., 2013c)

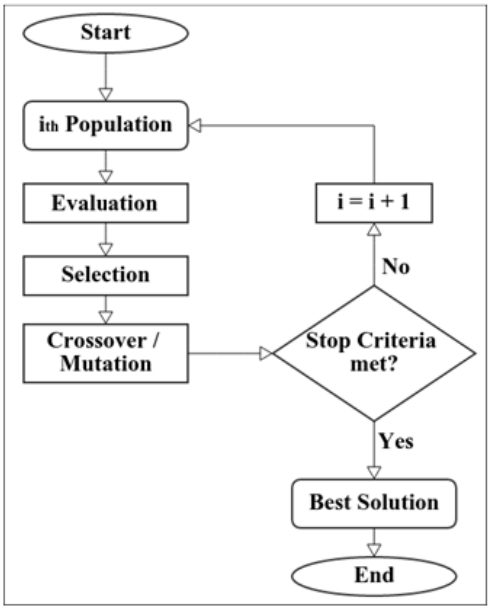
In case of stacking Denoising Autoencoder, the input corruption is only used for the initial denoising-training of each individual layer, so that it may learn useful feature extractors. Once the mapping has been learnt, it will henceforth be used on uncorrupted inputs. In particular no corruption is applied to produce the representation that will serve as clean input for training the next layer.

5 EVOLUTIONARY ALGORITHMS

Evolutionary Algorithms (EA) are general population-based metaheuristics optimization algorithms based on a direct analogy to Darwinian natural selection and genetics in biological system (Mitchell, 1998). The core aspect of EA is based on the concept of survival of the fittest in a population. Candidate solutions to the optimization problem play the role of individuals in a population, and the fitness function determines the quality of the solutions. Evolution of the population then takes place after the repeated application of biologically inspired operators, such as reproduction, mutation, recombination, and selection. Thus, EA perform a selective exploration of the search space using the fitness value of each individual that determine which is going to be the next point to be sampled (Holland, 1975). One of the main advantages of EA is their generality, i.e., they can be used in a broad range of conditions due to their simplicity and independence of the underlying domain of the problem. In this sense, only the codification of the population and the fitness function depends on the specific problem, the rest of the operators are (almost) independent of it (Rojas, 1996). Moreover, associated with the characteristics of exploration (the process of visiting new regions of a search space) and exploitation (the process of visiting those regions of a search space within the neighborhood of previously visited points), EA are capable of dealing with large search spaces efficiently (Mitchell, 1998).

Genetic Algorithms (GAs), one of the most popular EA methods, were first introduced by Holland (Holland, 1975). GA operate by modifying a set of candidate solutions (population) according to operators such as recombination and mutation. These operators are successively applied to the population in a loop, being each run of the loop called generation. The main idea is that these individuals evolve, tending to generate better ones, until acceptable results are obtained, or some stop condition is met. At the end, the fittest chromosome found during all generations is the GA's answer to the given problem. Figure 5.1 presents the logic of a standard GA. This method has been widely used in different tasks, including numerical and combinatorial optimization, multi-agent systems, economics, and bioinformatics (Bramlette, 1991; Whitley et al., 1989; Vignaux and Michalewicz, 1991).

Figure 5.1 – Execution of a standard Genetic Algorithm



Source: Gonzalez (2018)

5.1 Covariance Matrix Adaptation Evolution Strategy (CMAES)

The Covariance Matrix Adaptation Evolution Strategy (CMAES) (Hansen and Ostermeier, 2001) is an Evolutionary Algorithm for difficult non-linear non-convex derivative-free black-box optimization problems in continuous domain. In CMAES, a covariance matrix describing correlations between decision variables is learned and adapted during the search to maximize the likelihood of generating successful solutions. The algorithm relies on normally distributed mutative steps to explore the search space while adjusting its mutation distribution to make successful steps from the recent past more likely in the future. It comes in many variants, e.g., with extensions for handling of fitness noise (Hansen et al., 2009) and multi-modality (Auger and Hansen, 2005). Here we describe what can be considered a baseline version, featuring non-elitist (μ, λ) selection, cumulative step-size adaptation (CSA), and two different types of covariance matrix updates, namely a rank-1 update based on an evolution path, and the so-called rank-μ update based on the survivors of environmental truncation selection.

The state of CMAES is given by the parameters $m \in \mathbb{R}^d$, $\sigma > 0$ and $C \in \mathbb{R}^{dxd}$ of its multivariate normal search distribution $N(m, \sigma^2 C)$, as well as by the two evolution paths $p_s, p_c \in \mathbb{R}^d$. Besides these parameters the algorithm has a number of tuning constants, e.g., the

sizes of parent and offspring population μ and λ, the leaning rates c_1 and c_μ, and the rank-based weights ω. All these constants are set to their default values (Hansen, 2016).

In the tth iteration of the algorithm, the CMAES samples λ points from a multivariate normal distribution $N(m_t, \sigma^2{}_t C_t)$, evaluates the objective function f at these points, and adapts the parameters C_t, m_t, and σ_t. For a minimization task, the λ points are ranked by function value such that $f(x_{1,t}) \leq f(x_{2,t}) \leq \ ... \leq f(x_{\lambda,t})$. The distribution mean is set to the weighted average $m_{t+1} = \sum_{i=1}^{\mu} \omega_i \, x_{i,t}$. The weights depend only on the ranking, not on the function values directly. This renders the algorithm invariant under order-preserving transformation of the objective function. Points with smaller ranks (i.e., better objective function values) are given a larger weight ω_i with $\sum_{i=1}^{\lambda} \omega_i = 1$. The weights are zero for ranks larger than $\mu < \lambda$, which is typically $\mu = \lambda/2$. The covariance matrix is updated using two terms, a rank-1 and a rank-μ update. For the rank-1 update, in order to exploit information of correlations between generations, a long-term average of the changes of m_t is maintained by using the evolution path p_c:

$$p_{c,t+1} = (1 - c_c)p_{c,t} + \sqrt{c_c(2 - c_c)\mu_{eff}}\,(m_{t-1} - m_{t)}/\sigma_t \tag{2}$$

The coefficient $\mu_{eff} = \sum_{i=1}^{\mu} \omega_i{}^2$ is the effective sample size given the weights. Note that $p_{c,t}$ is large when the algorithm performs steps in the same direction, while it becomes small when the algorithm performs steps in alternating directions. The rank-μ update, which uses the information within the population of one generation, estimates the covariance of the weighted steps $x_{i,t} - m_t, 1 \leq i \leq \mu$. Combining rank-1 and rank-μ update gives the final update rule for C_t:

$$C_{t+1} = (1 - c_c - c_1)C_t + c_1 p_{c,t+1} p_{c,t+1}^T + \frac{c_\mu}{\sigma_t^2} \sum_{i=1}^{\mu} \omega_i \, (x_{i,t} - m_t)(x_{i,t} - m_t)^T \tag{3}$$

The update of the global step-size parameter σ is based on the cumulative step-size adaptation algorithm (CSA). It measures the correlation of successive steps in a normalized coordinate system. The goal is to adapt σ such that the steps of the algorithm become uncorrelated. Under the assumption that uncorrelated steps are standard normally distributed, a

long-term average over the steps should have the same expected length as a χ-distributed random variable, denoted by $\mathrm{E}\{\chi\}$. The long-term average has the form:

$$p_{s,t+1} = (1 - c_s)p_{s,t} + \sqrt{c_s(2 - c_s)\mu_{eff}}\, C_t^{-1/2}(m_{t-1} - m_{t)}/\sigma_t \qquad (4)$$

with $p_{s,1} = 0$. The normalization by the factor $C_t^{-1/2}$ is the main difference between equations (2) and (4). It is important because it corrects for a change of C_t between iterations. Without this correction, it is difficult to measure correlations accurately in the un-normalized coordinate system. For the update, the length of $p_{s,t+1}$ is compared to the expected length $\mathrm{E}\{\chi\}$ and t is changed depending on whether the average step taken is longer or shorter than expected:

$$\sigma_{t+1} = \sigma_{t+1} exp\left(\frac{c_s}{d_s}\left(\frac{|p_{s,t+1}|}{\mathrm{E}\{\chi\}} - 1\right)\right) \qquad (5)$$

The CMAES is an attractive option for non-linear optimization, if "classical" search methods, e.g. quasi-Newton methods (BFGS) and/or conjugate gradient methods, fail due to a non-convex or rugged search landscape (e.g. sharp bends, discontinuities, outliers, noise, and local optima). Moreover, the CMAES overcomes typical problems that are often associated with evolutionary algorithms:

- Poor performance on badly scaled and/or highly non-separable objective functions. Equation (3) adapts the search distribution to badly scaled and non-separable problems.
- The inherent need to use large population sizes. A typical reason for the failure of population-based search algorithms is the degeneration of the population into a subspace. This is usually prevented by non-adaptive components in the algorithm and/or by a large population size (considerably larger than the problem dimension). In the CMAES, the population size can be freely chosen, because the learning rates c_1 and c_μ in (3) prevent the degeneration even for small population sizes, e.g. $\lambda = 9$. Small population sizes usually lead to faster convergence, large population sizes help to avoid local optima.
- Premature convergence of the population. Step-size control in Equation (5) prevents the population to converge prematurely. However, it does not prevent the search to end up in a local optimum.

Therefore, the CMAES is considered as state-of-the-art and one of the most powerful EAs for real-valued optimization (Hansen and Ostermeier, 2001; Hansen et al., 2003; Hansen and Kern, 2004; Loshchilov et al., 2013) with many successful applications in real-world applications (Hansen, 2009). In machine learning, it has been used for direct policy search in reinforcement learning and hyperparameter tuning in supervised learning (Gomez et al., 2008; Heidrich-Meisner and Igel, 2009; Igel, 2010).

6 METHODOLOGY

This chapter examines thoroughly the basic definitions and concepts of time series analysis, assumptions, conditions, and principles. Considering these concepts, the experimental procedure involved in this work is also explained in this chapter.

6.1 Experimental Setup

Before the learning process can be launched, it is necessary to perform data partitioning. Data are divided into three sets: training set, validation set and testing set (Hyndman and Athanasopoulos, 2018). Usually, training set is the largest and it contains the data that will be used for training the model. Validation set provides an unbiased evaluation of a model fit on the training set. Moreover, it is also used to deal with the overfitting problem, which occurs when the model perfectly fits the training set, but it will have low performance on the newly observed data. The forecasting algorithms are trained on the training data, but the error is calculated for the validation set. Training is performed up to the moment when the error for validation set starts to increase; this is a regularization technique called early stopping. Overfitting and generalization issues should be kept in mind when comparing the models. Because of this, the performance on the test data is also considered during model comparison instead of just performance on the training set. The test set typically comprises about 20% of the last part of the total sample length although this value may change on how far ahead the forecasts are needed and on how long the sample is.

In this work, the training and validation sets are used by the CMAES algorithm to perform the selection of hyperparameters for each model in each dataset. It was used a fixed parametrization for the internals parameters of CMAES across all experiments. The population size (λ) is set to 24, and thus the first 24 solutions are sampled from the prior isotropic (not yet adapted) Gaussian with a mean of 0 and standard deviation of 0.5. The termination criteria are defined by a maximum number of generations ($g_{max} = 150$) and by a minimum fitness function target (f_{stop}), which depends on the dataset. MSE over the validation set is used as fitness function to evaluate each individual of the population. Table 6.1 presents the list of hyperparameters set to be optimized for each model.

Table 6.1 – Hyperparameters optimized in each model

Parameter	*Range*
Number of hidden layers	[1, 3]
Number of units per hidden layer	[8, 512]
Hidden layer activation function	[linear, tanh, relu]
Hidden layer dropout rate	[0, 0.5]
Number of training epochs	[50, 500]
Batch size	[32, 1024]
Optimization function	[rmsprop, adagrad, adadelta, adam]
Learning rate	[0.0001, 0.01]

Source: Gonzalez (2018)

After finishing the hyperparameters optimization process, a 5-fold rolling window cross-validation (Zivot and Wang, 2013) is performed using the training and validation set combined in order to access the performance of the best set of hyperparameters found. In this procedure, there is a series of test sets, each consisting of multiple observations. The corresponding training set consists only of observations that occurred prior to the observations that form the test set. Thus, no future observations can be used in constructing the forecast model. The forecast accuracy is computed by averaging over the test sets. Finally, it is performed a one-step-ahead prediction for the testing set in order to estimate the performance of each optimal model in unseen data.

6.2 Error and Performance metrics

Error and performance metrics are involved to calculate the difference between the predicted and target value, and respectively to measure the performance of the forecasting system. Forecasting accuracy is a reverse measure to the measure of forecasting error, which expressed as a deviation of predicted value ($\widehat{Z}_t$) and actual value (Z_t). There are several metrics to measure the forecasting error, each of them expresses a little bit different information. The following error metrics are evaluated in this work:

- Mean Absolute Error (MAE)

$$MAE = \frac{1}{N}\sum_{t=1}^{N}\left|Z_t - \widehat{Z}_t\right| \tag{6}$$

- Mean Squared Error (MSE)

$$MSE = \frac{1}{N}\sum_{t=1}^{N}\left(Z_t - \widehat{Z}_t\right)^2 \tag{7}$$

- Root Mean Squared Error (RMSE)

$$RMSE = \sqrt{\frac{1}{N}\sum_{t=1}^{N}\left(Z_t - \widehat{Z}_t\right)^2} \tag{8}$$

All error metrics are negatively-oriented scores, which means lower values are better. The suitability of MSE and RMSE measures is quite similar. Taking the square root of the average squared errors has some interesting implications for RMSE. Since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors. This means the RMSE should be more useful when large errors are particularly undesirable. MAE measures the average magnitude of the errors in a set of predictions, without considering their direction. It's the average over the test sample of the absolute differences between prediction and actual observation where all individual differences have equal weight. RMSE has the benefit of penalizing large errors more so can be more appropriate in some cases, for example, if being off by 10 is more than twice as bad as being off by 5. But if being off by 10 is just twice as bad as being off by 5, then MAE is more appropriate (Tofallis, 2015).

On the other hand, performance measurements can be considered as the inverse of the corresponding error metric. The error can be seen as a negative performance. The above three metrics are estimates of variance of residuals, or non-fit, in the population. Thus, it is also analyzed the Coefficient of Determination (R^2) (Fox, 1961), which is a measure of how well the regression line represents the data. If the regression line passes exactly through every point on the scatter plot, it would be able to explain all of the variation. The further the line is away from the points, the less it is able to explain. In regression models, R^2 corresponds to the squared correlation between the observed outcome values and the predicted values by the model. The coefficient of determination is such that $0 < R^2 < 1$. R^2 should not be the main statistical measure of the predictive power of a model, since it is more a measure of overall fitness than of forecast accuracy. For example, assuming a model which produces forecasts that are exactly 20% of the actual values. In this case, the R^2 value would be 1 (indicating perfect correlation), but the forecasts are not close to the actual values.

6.3 Benchmark Datasets

The Deep Learning models were evaluated on different time series, i.e. two synthetic series and one real data series. The synthetic series were generated using the Mackey-Glass and Lorenz System functions. The real data represents the hourly energy demand of the New England zone, US, provided by ISO New England. To allow a fair comparison of the results, each dataset is splitted into a training and a testing set, which will provide a performance reference for comparing the DL models against the baseline model.

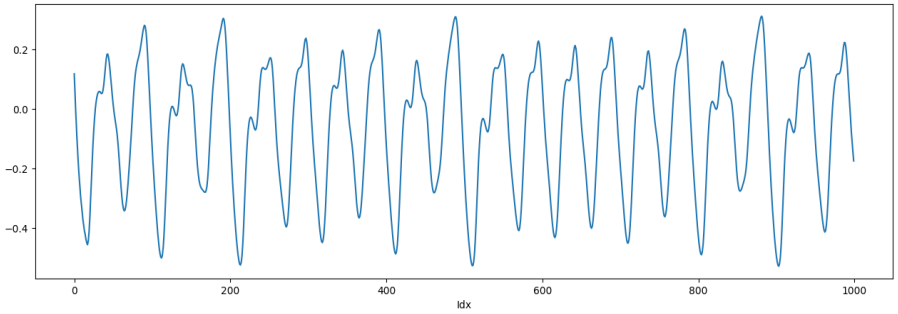
6.3.1 Mackey-Glass

The Mackey–Glass (MG) model (Mackey and Glass, 1977) describes the dynamics of physiological time delayed processes, mainly respiratory and hematopoietic (i.e. formation of blood cellular components), in which the actual evolution depends on the values of the variables at some previous times. This model exhibits a wide range of behaviors including periodic or chaotic solutions. The MG time series is generated by the Mackey–Glass nonlinear time-delay differential equation, which is described as follow:

$$x(t) = \frac{\beta x(t-\tau)}{1 + x(t-\tau)^{10}} - \gamma x(t) \tag{9}$$

where x (unitless) is the series in time t, and τ is the time delay. Here, it is assumed $\gamma = 0.1$, $\beta = 0.2$, $\tau = 17$ and $x(0) = 1.2$. Note that, if $\tau \geq 17$ the time series shows a chaotic behaviour (Doyne Farmer, 1982). This dataset is widely used in the literature as a benchmark for prediction models (Qiu et al., 2014; Prasad and Prasad, 2014; López-Caraballo et al., 2016). In this experiment, 5000 data points were sampled. The first 4000 data points were used as training set, and the remaining 1000 points as testing set. Figure 6.1 shows the first 1000 points of the training set.

Figure 6.1 – Mackey-Glass (MG) noiseless chaotic time series



Source: Gonzalez (2018)

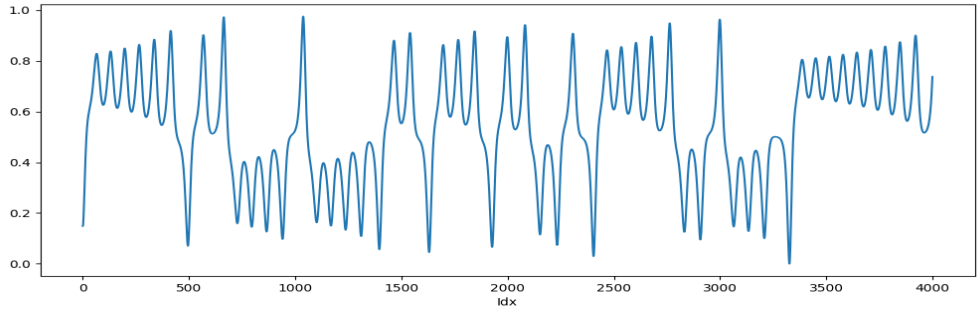
6.3.2 Lorenz System

Lorenz (Lorenz, 1963) proposed the Lorenz Attractor System in his attempt to model how an air current rises and falls while being heated by the sun. The Lorenz System shows how the state of a dynamical system (the three variables of a three-dimensional system) evolves over time in a complex, non-repeating pattern. The model is given by the following set of equations:

$$\begin{cases} \hat{x} = \sigma(x - y) \\ \hat{y} = x(\rho - z) - y \\ \hat{z} = xy - \beta z \end{cases} \tag{10}$$

The time series used in the experiments of this work is the x component of the Lorenz System, obtained by solving the differential equations system with initial conditions $\sigma = 10$, $\rho = 28$, $\beta = 8/3$, and $s_0 = (-13, -14, 47)$. The data was normalised to a maximum of 1. In this experiment, 8000 data points were sampled. The first 6400 data points were used as training set, and the remaining 1600 points as testing set. Figure 6.2 shows the first 4000 points of the training set.

Figure 6.2 – x component of the Lorenz System



Source: Gonzalez (2018)

The Lorenz System is well-known by its "butterfly effect" which indicates the sensitive dependence on initial conditions of chaos. Long-term prediction is almost impossible for this reason, however, short-term prediction, especially one-ahead prediction of chaotic time series can be realized if a predictor approximates the nonlinear system enough.

6.3.3 ISO New England hourly energy demand

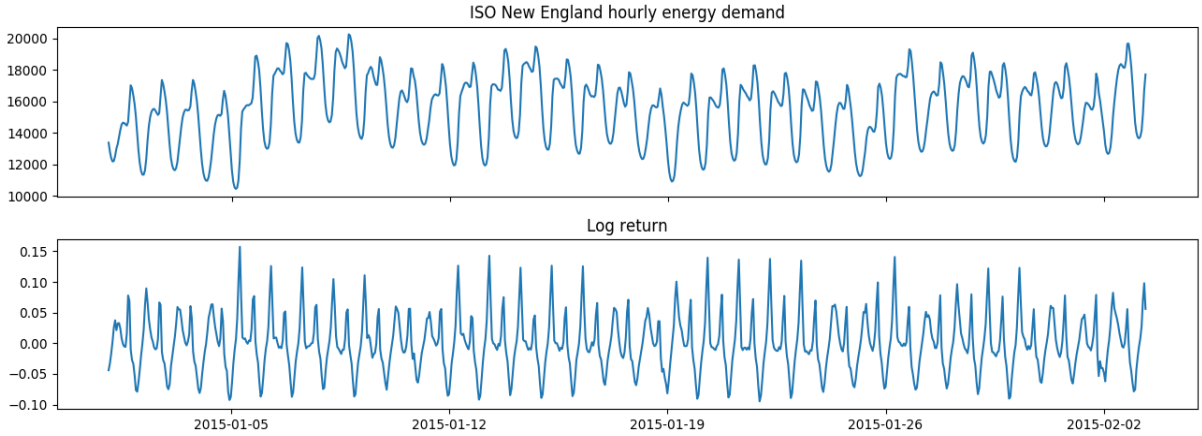
A booming economy is dramatically increasing electric loads in every industry and those associated with people's daily lives. Along with the fast development of electricity power market, decision-makers seek accurate load forecasting to set effective energy policies, such as those concerning new power plants and investment in facilities (Fan and Chen, 2006). Meeting the demand of all has become an important goal of electricity providers. Therefore, short-term load demand prediction is becoming important in such power systems. However, electricity load forecasting is challenging. Unfortunately, electric load data have various characteristics, including nonlinearity and chaos. Moreover, many exogenous factors interact with each other, affecting forecasting, such as economic activities, weather conditions, social activities, industrial production, and others. These effects increase the difficulty of load forecasting. (Hong et al., 2012; Souza et al., 2014).

In the last few decades, models for improving the accuracy of load forecasting have included the well-known Box–Jenkins' ARIMA model (Hussain et al., 2016), the Bayesian estimation model (Hippert and Taylor, 2010), and regression models (Wu et al., 2013). However, most of these models are theoretically based on assumed linear relationships between historical data and exogenous variables and so cannot effectively capture the complex nonlinear characteristics of load series, or easily provide highly accurate load forecasting. In order to improve the accuracy of load forecasting, many artificial intelligence (AI) approaches have

been used and been combined to develop powerful forecasting methods, such as artificial neural networks (ANNs) (Hernández et al., 2014; Ertugrul, 2016), support vector regression (SVR) (Hong, 2011; Fan et al., 2016), and evolutionary algorithms (Ghanbari et al., 2013; Bahrami et al., 2014). However, the shortcomings of these AI approaches include the need to determine the structural parameters, the time required for knowledge acquisition, and a lack of correct and consistent heuristic rules to generate a complete domain knowledge base. Busseti et al. also conducted simulations to compare deep learning methods with traditional shallow neural networks (Busseti et al., 2012). The work successfully showed the advantages of deep learning architectures to the problems of electricity load demand forecasting.

The third dataset used in this work consists of 17544h of load data from the 2017 Global Energy Forecasting Competition (GEFCom 2017), which ranges from 00:00 1 January 2015 to 23:00 31 December 2016 (Hong, 2017). The data represents the hourly energy demand of the New England zone, US, provided by ISO New England. The data set is divided into two subsets: a training set (16727 h of load data from 4 January 2015 to 30 November 2016), and a testing set (743 h of load data from 1 December 2016 to 31 December 2016). The data was normalized calculating the hourly log return. Figure 6.3 shows the original data (on top) and the transformed (logarithmic returns) version (on the bottom).

Figure 6.3 – Original and transformed version of the hourly energy demand data



Source: Gonzalez (2018)

6.4 Time Series Analysis

Time series analysis comprises methods or processes that breakdown a series into components and explainable portions that allows trends to be identified, estimates and forecasts

to be made. Basically, time series analysis attempts to understand the underlying context of the data points through the use of a model to forecast future values based on known past values.

Time series is different from more traditional classification and regression predictive modeling problems. The temporal structure adds an order to the observations. This imposed order means that important assumptions about the consistency of those observations needs to be handled specifically when modeling it. First, there are assumptions that the summary statistics of observations are consistent. In time series terminology, we refer to this expectation as the time series being stationary. Similarly, often a better model is possible if a correlation mechanism can be determined in the data. It is important to understand that correlations are useful for forecasting, even when there is no causal relationship between the two variables, or when the correlation runs in the opposite direction to the model. In this section these two assumptions are better explained and studied for each one of the three datasets used in this work.

6.4.1 Stationarity

A stochastic process is said to be stationary if static properties like mean and variance do not vary over time, i.e. if it is free of trends, shifts and periodicity. Otherwise a process is called non-stationary, which provides a drift in the concepts that a model may try to capture. Stationary processes are much easier and lots of analysis are done for them, while in practice we meet non-stationary processes more often. A stationary time series will return to its long-term mean after a random shock, which means it is possible to reliably forecast the time series (Ross, 2014).

Stationarity is used as a tool in time series analysis, where the raw data are often transformed to become stationary. For example, economic data are often seasonal or dependent on a non-stationary price level. We need first identify whether or not the time series under study is stationary; if not, transformation or differencing is needed to make the time series stationary. To determine whether the time series is stationary or not, a plot of the series is the first step to see if the mean and the variance remain the same through the time. Additional statistical tests can then be performed to inform the degree to which a hypothesis that the data is nonstationary can be rejected or not.

The Augmented Dickey-Fuller (ADF) (Greene, 2003) test is a type of statistical test called a unit root test. The intuition behind a unit root test is that it determines how strongly a

time series is defined by a trend. The null hypothesis of the test is that the time series can be represented by a unit root, that it is not stationary (has some time-dependent structure). The alternative hypothesis (rejecting the null hypothesis) suggests the time series does not have a unit root, meaning it is stationary. The results are interpreted using the p-value from the test. Usually 5% threshold is being used, which means that the null-hypothesis is rejected if the p-value is less than 0.05. Table 6.1 shows the results of the ADF test for each dataset. Along with the p-value, it is shown the test statistic (the more negative this value, the more likely the data is stationary) and the critical values for three levels of significance (if the test statistic is more negative than a given critical value, this suggests that we can reject the null hypothesis with a significance level of less than the given level). The analysis of this table suggests that we can reject the null hypothesis with a significance level of less than 1% for all the datasets, meaning that these time series are stationary.

Table 6.1 – ADF test results for each dataset

	Mackey-Glass	*Lorenz*	*Energy demand*
Test Statistic	-15.00006	-5.76192	-23.14648
Critical Value (1%)	-3.43166	-3.43117	-3.43072
Critical Value (5%)	-2.86212	-2.8619	-2.8617
Critical Value (10%)	-2.56707	-2.56696	-2.56685
p-value	1.09313e-27	5.64626e-07	0.0

Source: Gonzalez (2018)

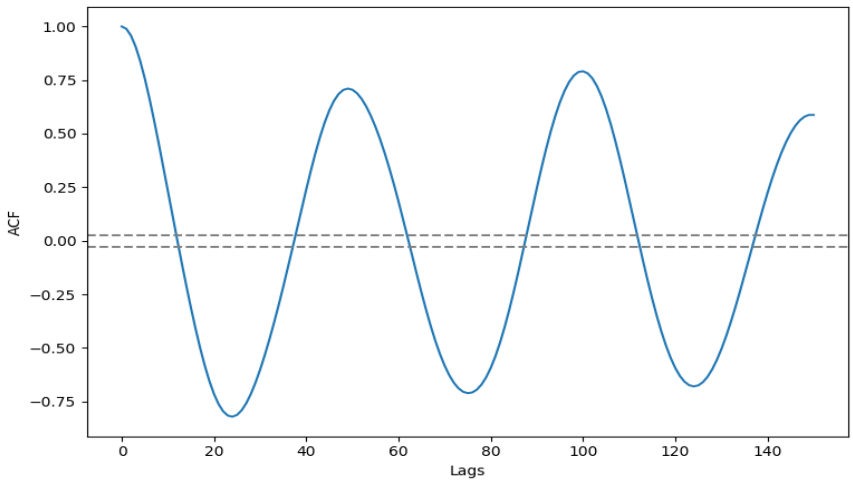
6.4.2 Autocorrelation

Dependencies between the actual and historical values represent a fundamental principle of time series forecasting. It can be easily observed, that each value of the series is very similar to its neighboring values. Additionally, some time series present a seasonal component, which means, that each value is also dependent on the values of identical time, but one season ago. Formally, any statistical dependency between two entities is denoted as a correlation, and is expressed by a corresponding coefficient (Dietrich, 2017).

Since the correlation of the time series observations is calculated with values of the same series at previous times, this is called a serial correlation, or an autocorrelation. The autocorrelation function calculates the correlations between the time series and its shifted (lagged) copies at different points in time. The autocorrelation coefficients are usually calculated for a specific range of lags and are expressed in the form of graph called correlogram,

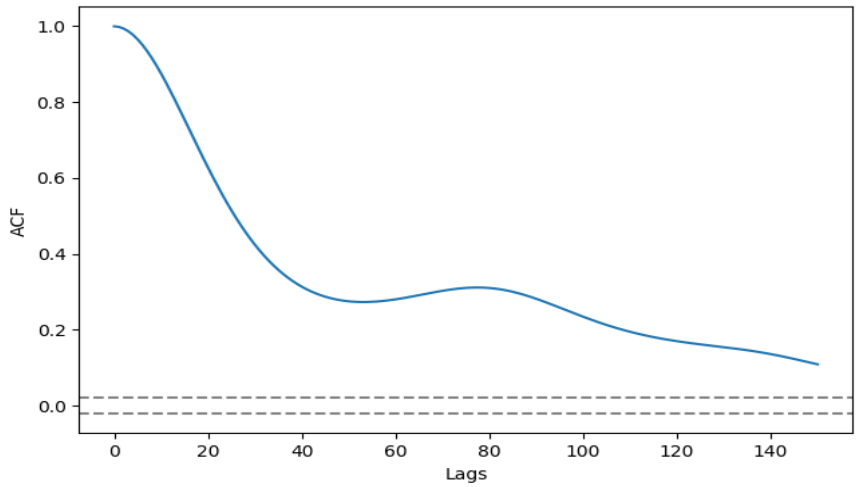
a visual representation of the AutoCorrelation Function (ACF). The autocorrelation with lag zero always equals 1, because this represents the autocorrelation between each term and itself. Confidence intervals are also drawn. By default, this is set to a 95% confidence interval, suggesting that correlation values outside of these lines are very likely a correlation and not a statistical fluke (Filliben, 2013). The following figures presents the ACF plot for the three datasets used in this work. Figure 6.4 shows the ACF for the Mackey-Glass data. Based on this chart, it is possible to infer that the data has some seasonality, since there is decay and then a spike in the coefficients at regular intervals. On the other hand, the ACF for the Lorenz System dataset, presented in Figure 6.5, decays more slowly (i.e., has significant spikes at higher lags) and does not present any seasonal pattern. Finally, Figure 6.6 shows the autocorrelations associated with the New England hourly energy demand data. It is possible to observe that the data is also characterized by a seasonal trend, in this case a daily trend. Hourly energy demand is often correlated with the same time of the previous day, which explains the spikes at lag 24, 48, and so on.

Figure 6.4 – Mackey-Glass ACF



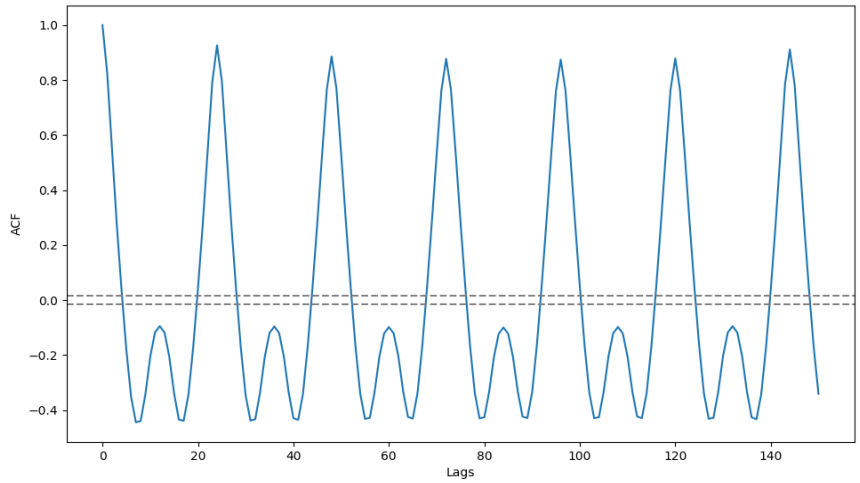
Source: Gonzalez (2018)

Figure 6.5 – Lorenz ACF



Source: Gonzalez (2018)

Figure 6.6 – Original and transformed version of the hourly energy demand data



Source: Gonzalez (2018)

A univariate time series dataset is only comprised of a sequence of observations. Therefore, it must be transformed into input features and output features in order to use supervised learning algorithms. The problem of time series prediction is merely a problem of extracting a manageable set of good features along the temporal dimension of the data. Temporal feature selection depends strongly on the availability of features in their temporal relation to outputs as well as the depth of outputs prediction. To take full advantage of these characteristics the prediction problem has to be considered within an appropriate temporal prediction paradigm. One benefit to autocorrelation is that we can identify patterns within the time series, which helps in determining seasonality, the tendency for patterns to repeat at

periodic frequencies. In that sense, ACF coefficients are used in order to identify the most informative set of lags to be used as input features for the forecasting models used in this work.

6.5 Models Implementation

This project was developed using the programing language Python and several packages available for it. Created by a former Google employee Guido van Rossum, Python is said to be very user friendly and is one of the most versatile languages with applications ranging from scripting macros to creating complex cloud-based systems (Nelli, 2015). Along with all its packages, Python is often considered a wild card for those who want to perform statistical and data analysis. The ability to interface with other languages, such as C and Fortran as well as develop data analysis projects integrated to Web Servers and internet through support libraries makes Python unique among similar languages, such as R and MATLAB.

Some Python packages were very useful for the development of this project. In order to implement the Deep Learning models, it was used the *Keras* package (Chollet, 2015), which is a high-level neural networks API that focuses on being user-friendly, modular, and extensible. The models developed using *Keras* run on top of *TensorFlow* (Abadi et al., 2015), which is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. Finally, the package *cma* (Hansen et al., 2018) provides a simple interface for applying the CMAES algorithm to optimize the hyperparameters of the selected deep learning models, as it makes creating customized objective functions possible.

7 EXPERIMENTAL RESULTS

In this chapter, it is presented the results and discussed the outcome of the experiments. It is shown the performance of deep learning models when applied in time series datasets and also evaluated some of their key parameters to obtain some insights about their suitability for the domain. Three deep learning architectures are studied: Stacked Autoencoder (SAE), Denoising Stacked Autoencoder (SDAE), and Recurrent Neural Network with LSTM cells (LSTM). A standard Multi-layer Neural Network (MLP) is employed as baseline. The performance of the proposed forecasting models was tested with three different time series (which are described in section 6.3). A comparison of the short-term (one-step ahead) forecasting precision of each proposed model is presented separately for each time series problem. Different error and performance measurements, as described in section 6.2, are reported in order to compare the results for the training and testing datasets.

In general, after fitting a time series model the residuals should be white noise (Kleijnen, 1986). By definition, a time series is considered white noise if the variables are normally, independently, and identically distributed with zero mean. Thus, they should have no autocorrelation. Therefore, if the ACF plot of the residuals shows significant autocorrelation and not just one spike at high lag order this maybe an indicate that the residuals are not white noise, and thus the model was not correctly specified (e.g. the lag order is not sufficient). In this case, there is some information leftover which should be accounted for in the model in order to obtain better forecasts. The forecasts from a model with autocorrelated errors are still unbiased, and so are not “wrong”, but they will usually have larger prediction intervals than they need to. In that sense, in order to further analyze the forecasting performance of the proposed models, it is also evaluated the ACF plot of the residuals for each data set.

Designing an appropriate DNN architecture is challenging and it is one of the main obstacles in exploiting DNNs in practice. As a powerful evolutionary optimization tool, CMAES is applied to optimize the hyperparameters set of each model. As presented in the following sections, CMAES significantly boosted the forecasting performance of all models. Also, we report the number of generations after which CMAES converged to the considered optimal (best) DNN hyperparameters values in which they were not further improved.

7.1 Mackey-Glass

This section examines how efficient the proposed models are relative to forecasting a time series that is generated by the Mackey-Glass differential delay equation. The forecasting horizon is one step-ahead. After optimizing the hyperparameters of each model, a 5-fold rolling window cross validation step is performed over the training set for all methods, aiming to better estimate the training performance. However, just as the overfitting problem exists when forecasting time series data, a good learning (training) result may not mean a robust predictor. Therefore, it should be also analyzed the performance of each model in unseen data.

The results presented in Table 7.1 refer to the mean and standard deviation in the performance of each model at the training dataset. The best results are highlighted. In a general analysis of this table, it is possible to state that all models presented great forecasting performance. However, LSTM obtained the best performance taking into consideration all the error measures. Moreover, it is worth to note that both models with pre-training (SAE and SDAE) have obtained the worst performance. It is also important to analyze the standard deviation of the cross-validation process, because it gives insights about the model stability. Thus, given that the LSTM has the smallest standard deviation for all metrics, it can be said that it is also the most stable model.

Table 7.1 – CV results for Mackey-Glass training dataset

	MLP	*SAE*	*SDAE*	*LSTM*
R^2	0.9980 ± 8.8e-4	0.9972 ± 1.2e-3	0.9979 ±4.9e-4	0.9987 ± 4.8e-4
MSE	9.3e-5 ± 3.8e-5	1.2e-4 ± 5.7e-5	9.8e-5 ± 2.3e-5	5.9e-5 ± 2.2e-5
RMSE	9.4e-3 ± 1.8e-3	1.1e-2 ± 2.5e-3	9.8e-3 ± 1.7e-3	7.5e-3 ± 1.4e-3
MAE	7.8e-3 ± 1.2e-3	9.2e-3 ± 2.2e-3	8.1e-3 ± 9.7e-4	6.2e-3 ± 1.1e-3

Source: Gonzalez (2018)

In order to analyze the forecasting performance over unseen data, Table 7.2 shows the results obtained for the test set. LSTM also presented the best generalization ability, and outperformed all the other methods. LSTM obtained a RMSE 35.6% smaller than MLP, 47.1% smaller than SAE, and 36.5% smaller than SDAE. Furthermore, as analyzed in the training phase, SAE and SDAE also presented the worst performance in general.

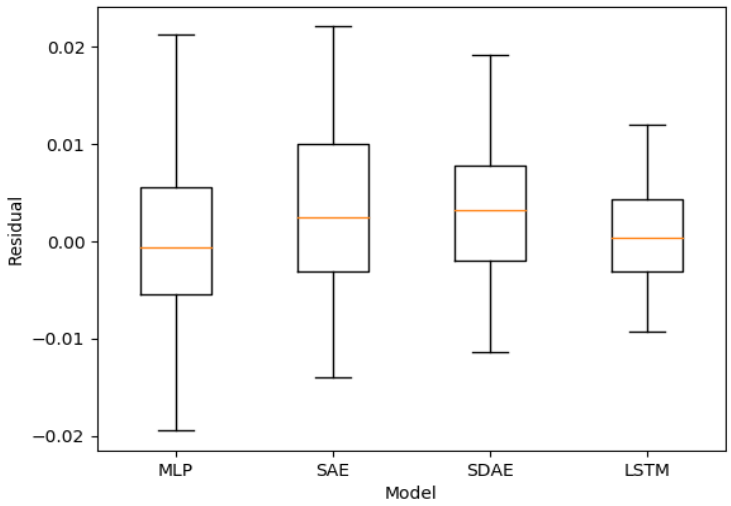
Table 7.2 – Forecasting performances for Mackey-Glass test set

	MLP	*SAE*	*SDAE*	*LSTM*
R^2	0.9988	0.9983	0.9988	0.9995
MSE	5.4e-5	7.9e-5	5.6e-5	2.2e-5
RMSE	7.3e-3	8.9e-3	7.4e-3	4.7e-3
MAE	5.9e-3	7.2e-3	6.1e-3	3.9e-3

Source: Gonzalez (2018)

The box plots of the residuals for the four models at the testing set is shown in Figure 7.1. It can be observed that the box plot for LSTM is narrower than those of MLP, SAE and SDAE, indicating that the LSTM residuals are less spread out as compared to the other models. The box plots of LSTM and MLP also suggest that the residuals medians are much nearer to 0 than SAE and SDAE. However, since the box plot for MLP has a greater variance than the one for LSTM, it is possible to state that the LSTM residuals are the most symmetrically distributed around the zero value. Finally, SAE and SDAE residuals are pulled out (skewed) towards the top of the plot (positive values), which means that theirs forecasted values were too low.

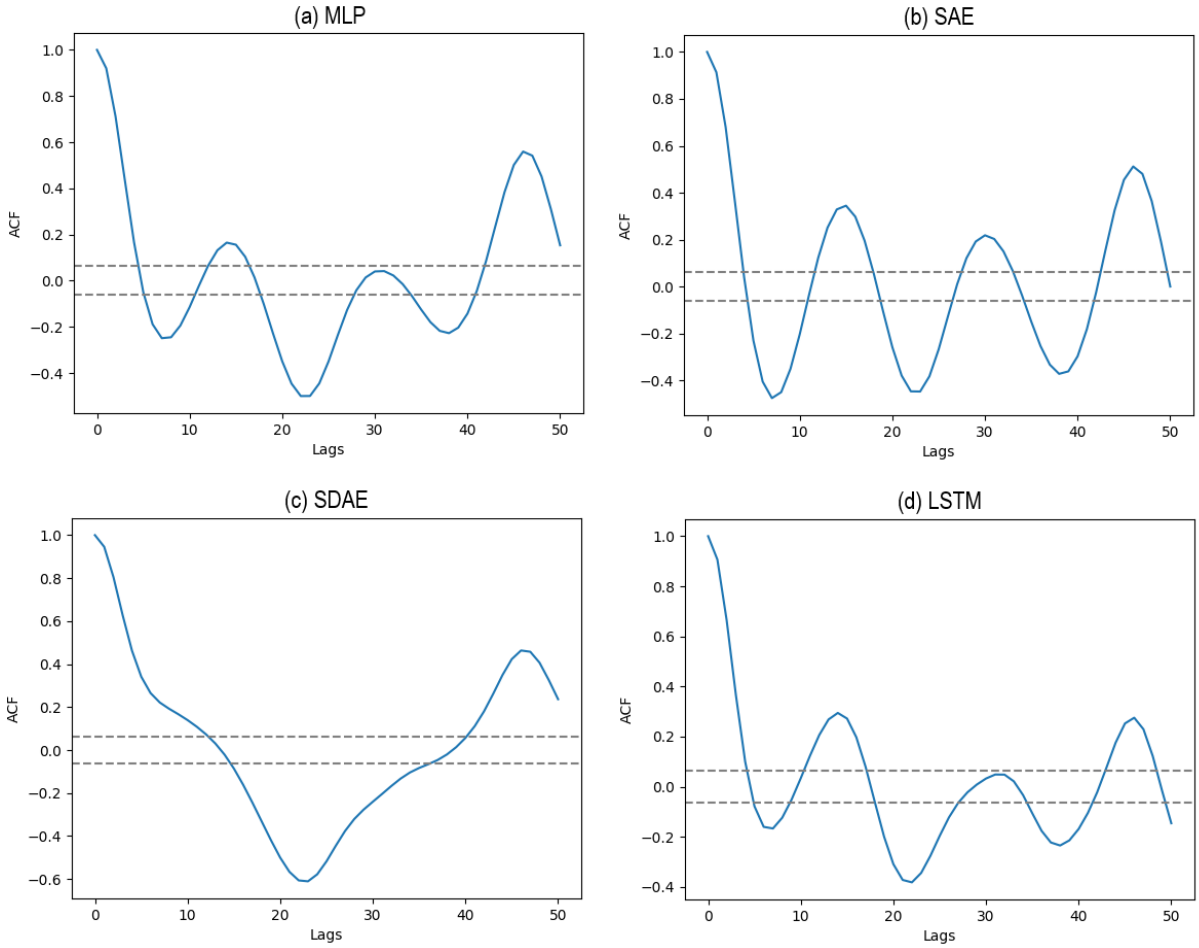
Figure 7.1 – Box plots of the residuals for Mackey-Glass test set



Source: Gonzalez (2018)

Aiming to further analyze the forecasting results, the ACF plots of the residuals for each model is shown in Figure 7.2. Based on these plots, it is possible to state that all the four residuals series are not random white noise, since they all present considerable autocorrelations spikes. Therefore, despite all models have presented good forecasting performance in the testing set, they still have room for improvement.

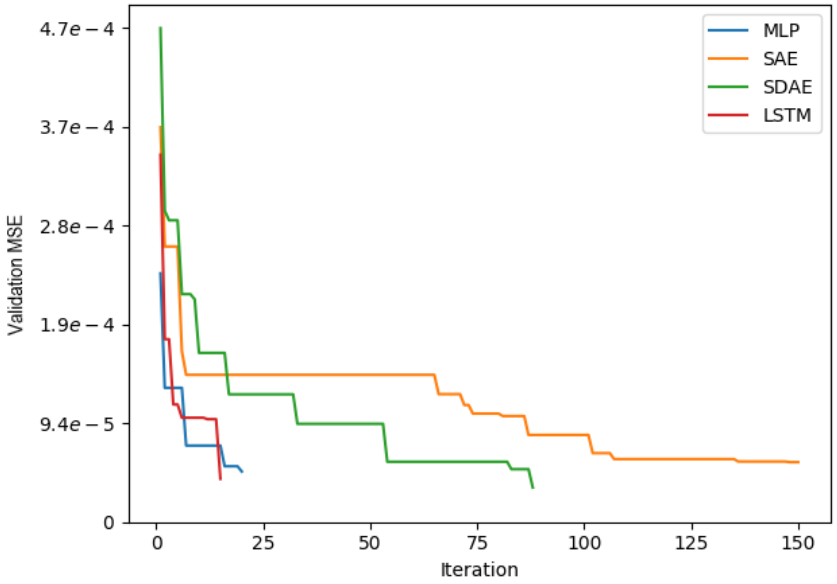
Figure 7.2 – ACF plots of the residuals for Mackey-Glass test set



Source: Gonzalez (2018)

Figure 7.3 shows the convergence of the validation MSE of each model when optimized by the CMAES algorithm for a maximum of 150 iterations or until a target value (f_{stop}) is reached. For the Mackey-Glass dataset, f_{stop} is set to 5e-5. It is possible to see that CMAES is able to perform an important error reduction on the initial population, which indicates the effectiveness of the algorithm on hyperparameters selection and the use of misclassification error as a guide to lower error in the solution space. LSTM and MLP presented the fastest MSE reduction over time, reaching the termination target value in 15 and 20 iterations respectively. The optimization process for the SDAE got terminated after 88 iterations when it reached the f_{stop} value. CMAES do not work that well on optimizing SAE, since it didn't reach the f_{stop} value and got terminated after the maximum number of iterations.

Figure 7.3 – Validation MSE by different methods using CMAES algorithm in Mackey-Glass dataset



Source: Gonzalez (2018)

Finally, Table 7.3 shows the best set of hyperparameters found for each model for the Mackey-Glass dataset.

Table 7.3 – Best hyperparameters sets found for Mackey-Glass dataset

Parameter	*MLP*	*SAE*	*SDAE*	*LSTM*
Number of hidden layers	1	3	3	2
Number of units per hidden layer	[392]	[8, 282, 252]	[368, 58, 175]	[487, 305]
Hidden layer activation function	[linear]	[linear, linear, linear]	[linear, linear, linear]	[tanh, tanh]
Hidden layer dropout rate	[0.0104]	[0.0334, 0.4162, 0.0409]	[1.6e-7, 0.068, 0.058]	[1.2e-4, 0.459]
Number of training epochs	393	156	494	365
Batch size	32	42	365	414
Optimization function	rmsprop	rmsprop	adam	adam
Learning rate	2.1e-4	1.6e-4	1.4e-4	5.6e-3

Source: Gonzalez (2018)

7.2 Lorenz System

In the second numerical example, the x component of the Lorenz System is used to study. The relevant modeling procedures are as in the previous section. Table 7.4 presents the performance results of the rolling window CV by each model. Similar to the previous example, all models achieve very low training errors. However, for this dataset, the standard MLP outperformed all the other three deep learning models. Again, both SAE and SDAE achieved the highest learning errors. Regarding the CV standard deviations, MLP also presented the smallest values, which indicates an expected small variance in the generalization error.

Table 7.4 – CV results for Lorenz System training dataset

	MLP	*SAE*	*SDAE*	*LSTM*
R^2	0.9996 ± 1.5e-4	0.9934 ± 6.8e-3	0.9985 ± 1.0e-3	0.9994 ± 3.0e-4
MSE	1.5e-5 ± 7.4e-6	2.8e-4 ± 3.0e-4	5.6e-5 ± 4.1e-5	2.4e-5 ± 1.4e-5
RMSE	3.8e-3 ± 1.0e-3	1.3e-2 ± 9.5e-3	7.0e-3 ± 2.5e-3	4.7e-3 ± 1.3e-3
MAE	2.8e-3 ± 8.6e-4	1.2e-2 ± 9.4e-3	5.4e-3 ± 2.6e-3	3.3e-3 ± 9.2e-4

Source: Gonzalez (2018)

Table 7.5 lists all the values of the metrics for the out-of-sample forecasting. Here, out-of-sample forecasting means forecasting with testing data. As expected, MLP presented the best overall forecasting performance over the test set. However, SAE achieved a very similar performance, outperforming both SDAE and LSTM. Regarding only the RMSE values, MLP obtained a RMSE slightly smaller (6.25%) than SAE, and much smaller than SADE and LSTM (47.3% and 57.1% respectively). Furthermore, differently from the training results, LSTM presented the worst performance.

Table 7.5 – Forecasting performances for Lorenz System test set

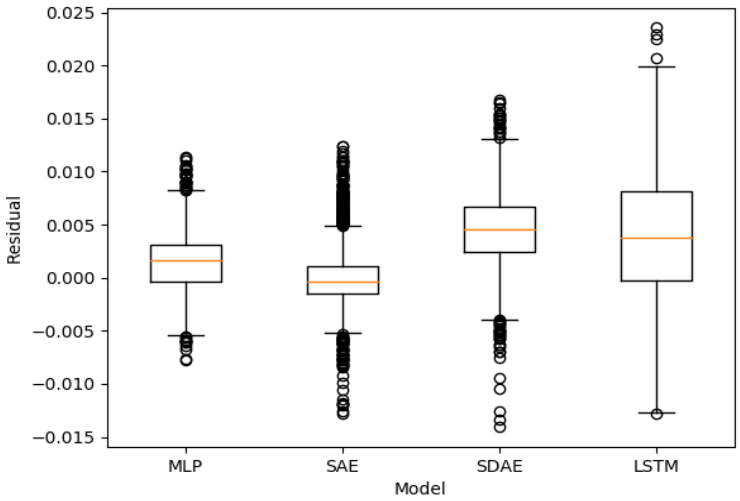
	MLP	*SAE*	*SDAE*	*LSTM*
R^2	0.9997	0.9997	0.9992	0.9988
MSE	9.1e-6	1.0e-5	3.2e-5	4.9e-5
RMSE	3.0e-3	3.2e-3	5.7e-3	7.0e-3
MAE	2.3e-3	2.2e-3	4.9e-3	5.7e-3

Source: Gonzalez (2018)

The distribution of the testing residuals can be seen in Figure 7.4. The box plot of the MLP indicates that it is the most symmetrically distributed around the median value. SAE

presents the median value closer to zero, but it also has more outliers. It can be observed by the box plot of LSTM and SDAE that these two models have the greater variance in the residual values.

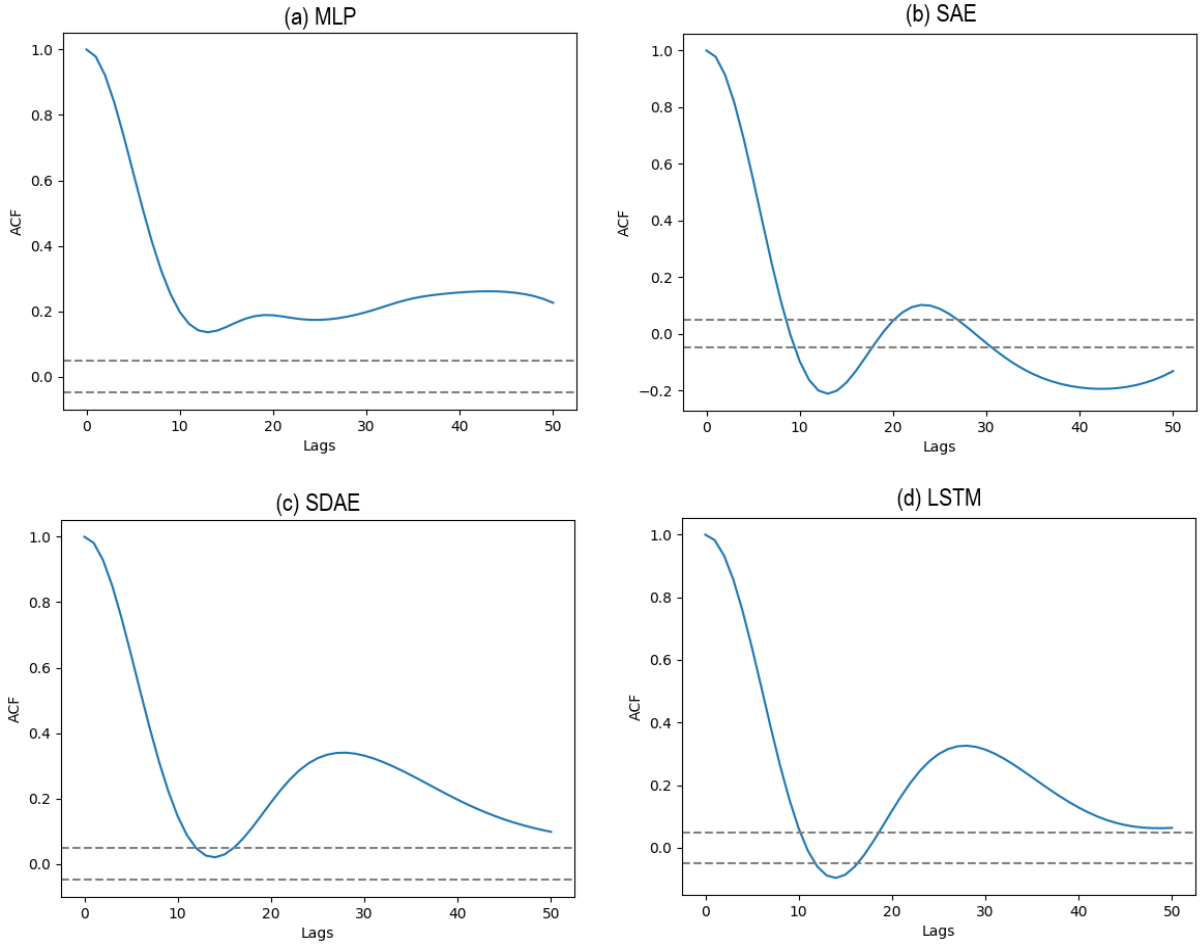
Figure 7.4 – Box plots of the residuals for Lorenz System test set



Source: Gonzalez (2018)

Figure 7.5 presents the ACF plots of the residuals for each model. Similar to the previous example, all the four residuals series cannot be classified as white noise. Comparing the MLP and SAE plots, it is possible to say that SAE presents residuals with less autocorrelation, which means that it was able to better model the data. The ACF plots for LSTM and SDAE are very similar, both showing spikes around the lag number 25. Again, despite all models have shown good forecasting performance, the residuals are not random, meaning that the models don't have enough information to correctly make predictions.

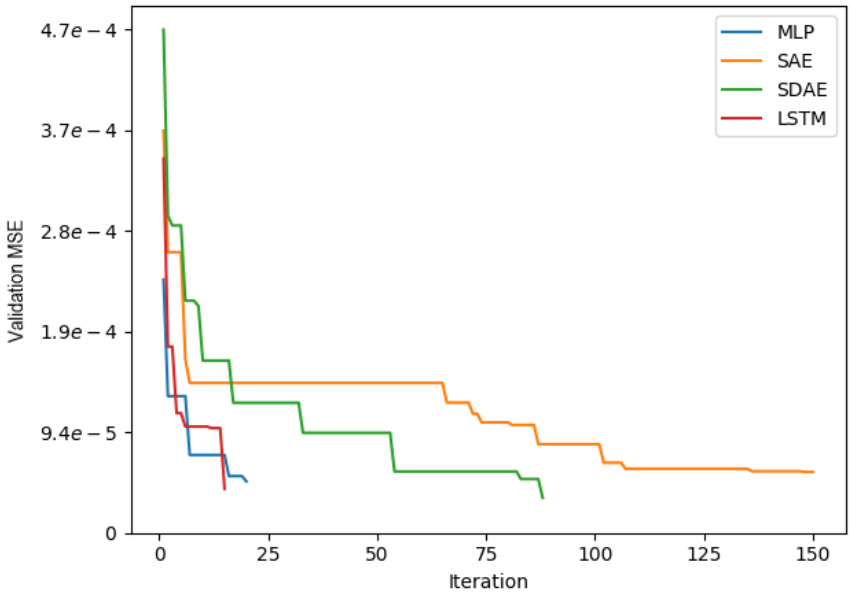
Figure 7.5 – ACF plots of the residuals for Lorenz System test set



Source: Gonzalez (2018)

By analyzing Figure 7.6, it is possible to note that CMAES steadily improved the validation error over time for all models. For the Lorenz System dataset, the target value (f_{stop}) is also set to 5e-5. MLP showed the fastest MSE convergence, reaching the target stop value after only 29 iterations. SAE took longer (128 interactions) to reach the f_{stop} limit. In the SDAE and LSTM cases, the CMAES was able to rapidly reduce the validation error on the initial populations, but it got trapped in a local optimal solution and no further improvement was achieved in the best solution.

Figure 7.6 – Validation MSE by different methods using CMAES algorithm in Lorenz System dataset



Source: Gonzalez (2018)

Table 7.6 shows the best set of hyperparameters found for each model for the Lorenz Systems dataset.

Table 7.6 – Best hyperparameters sets found for Lorenz System dataset

Parameter	*MLP*	*SAE*	*SDAE*	*LSTM*
Number of hidden layers	1	3	3	1
Number of units per hidden layer	[443]	[430, 235, 453]	[424, 31, 57]	[209]
Hidden layer activation function	[relu]	[tanh, linear, relu]	[relu, tanh, tanh]	[tanh]
Hidden layer dropout rate	[2.2e-3]	[0.082, 0.001, 0.009]	[6.9e-4, 0.031, 2e-4]	[3.7e-6]
Number of training epochs	492	483	473	365
Batch size	32	385	1018	33
Optimization function	adagrad	adam	adam	adagrad
Learning rate	9.9e-3	5e-4	7e-4	9.8e-3

Source: Gonzalez (2018)

7.3 ISO New England hourly energy demand

The third use case involves the ISO New England hourly electricity load data from 2017 Global Energy Forecasting Competition (GEFCOM 2017). It aims to study the forecasting

performance of each model over real data. The relevant modeling procedures are as in the preceding two examples. As can be seen in Table 7.7, LSTM achieved the best average performance across MSE, RMSE, MAPE, and R^2. SAE presented the worst training results for all the four metrics.

Table 7.7 – CV results for ISO New England hourly energy demand training dataset

	MLP	*SAE*	*SDAE*	*LSTM*
R^2	0.9512 ± 6.0e-3	0.8949 ± 2.6e-2	0.9508 ± 9.4e-3	0.9553 ± 1.6e-2
MSE	1.3e-4 ± 1.3e-5	2.8e-4 ± 8.0e-5	1.3e-4 ± 2.5e-5	1.1e-4 ± 3.7e-5
RMSE	1.1e-2 ± 6.0e-4	1.6e-2 ± 2.2e-3	1.1e-2 ± 1.0e-3	1.0e-2 ± 1.7e-3
MAE	8.1e-3 ± 2.8e-4	1.2e-2 ± 2.3e-3	8.3e-3 ± 7.2e-4	7.9e-3 ± 1.4e-3

Source: Gonzalez (2018)

Table 7.8 shows the results for the testing set. LSTM clearly outperforms all the other models regarding any metric. LSTM achieved a RMSE 42.14% lower than SAE, and 19% lower than MLP and SDAE. SAE also presented the worst generalization performance when considering all the metrics.

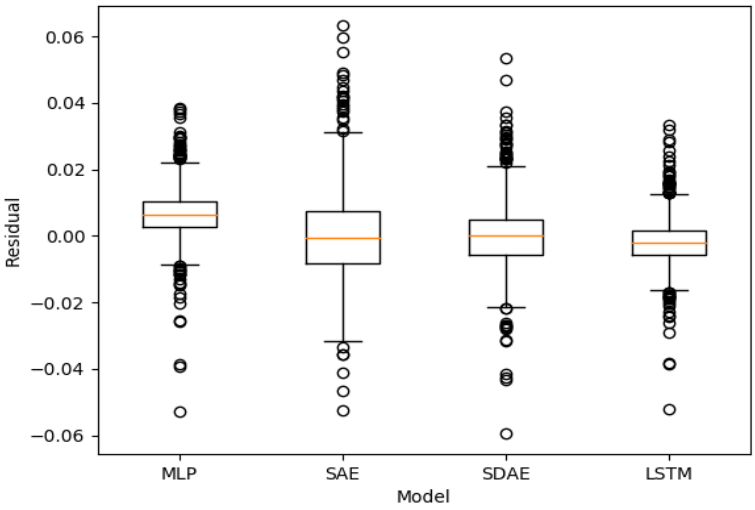
Table 7.8 – Forecasting performances for ISO New England hourly energy demand test set

	MLP	*SAE*	*SDAE*	*LSTM*
R^2	0.9508	0.9162	0.9512	0.9724
MSE	1.1e-4	2.0e-4	1.1e-4	6.7e-5
RMSE	1.0e-2	1.4e-2	1.0e-2	8.1e-3
MAE	8.6e-3	1.0e-2	7.6e-3	5.8e-3

Source: Gonzalez (2018)

Figure 7.7 shows the residuals box plots of each model at the testing set. Following the analysis of the previous two tables, it can be observed that LSTM residuals present the lowest variance and are the most symmetrically distributed around the median value. All the residuals medians are closer to zero. However, the SAE box plot presents the greatest variance and more outliers with positive values, which matches with the poor performance observed in the forecasting results in the test set presented in the previous table.

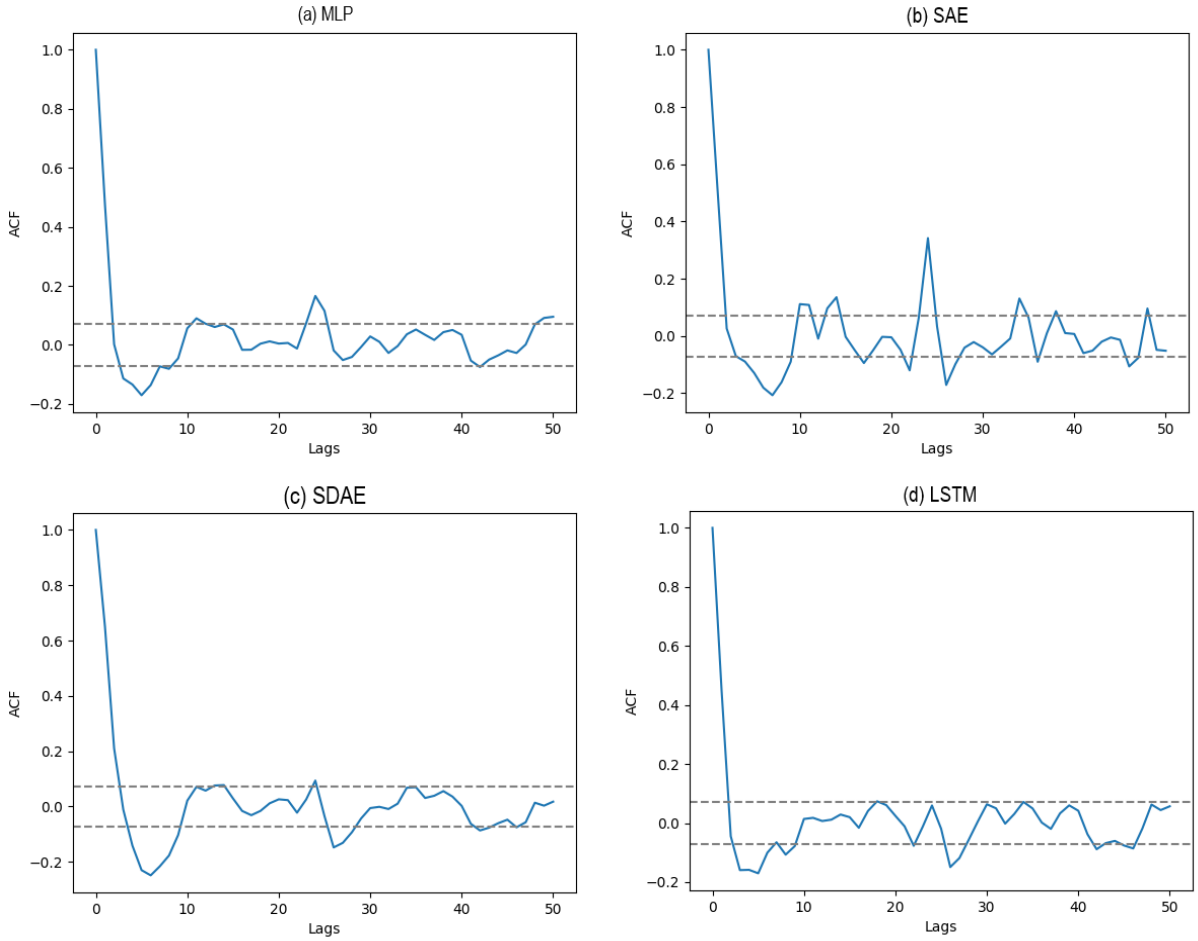
Figure 7.7 – Box plots of the residuals for ISO New England hourly energy demand test set



Source: Gonzalez (2018)

The ACF plots of the testing residuals for each model is shown in Figure 7.8. The LSTM autocorrelation plot shows that most of the spikes of each lag are not statistically significant, which indicates that the residuals are not highly correlated and do not show any particular pattern. On the other hand, the ACF plot of MLP, SAE and SDAE present a peak around lag 24. This means that LSTM was able to model the seasonal components of the hourly data better than the other models.

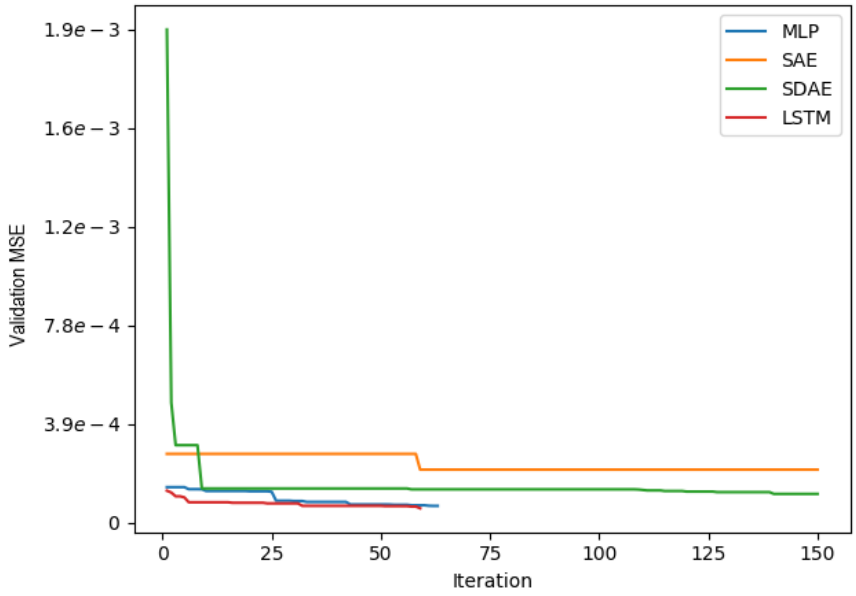
Figure 7.8 – ACF plots of the residuals for ISO New England hourly energy demand test set



Source: Gonzalez (2018)

Figure 7.9 shows the convergence of the validation MSE of each model when optimized by CMAES. For this dataset target value (f_{stop}) used as stop criteria is set to 1.5e-4. CMAES was able to perform an important error reduction on the initial population only for SDAE, whereas it was not so effective for the other models. Moreover, the validation MSE for SAE and SDAE did not present any considerable improvement over the time and did not reach the minimum target value, which indicates that CMAES could have got trapped around a local optimal solution. LSTM and MLP presented the fastest MSE reduction over time, reaching the f_{stop} value after 59 and 63 iterations respectively.

Figure 7.9 – Validation MSE by different methods using CMAES algorithm in ISO New England hourly energy demand dataset



Source: Gonzalez (2018)

Table 7.9 shows the best set of hyperparameters found for each model for the ISO New England hourly energy demand dataset.

Table 7.9 – Best hyperparameters sets found for ISO New England hourly energy demand dataset

Parameter	*MLP*	*SAE*	*SDAE*	*LSTM*
Number of hidden layers	1	3	3	2
Number of units per hidden layer	[392]	[8, 282, 252]	[368, 58, 175]	[508, 109]
Hidden layer activation function	[linear]	[linear, linear, linear]	[tanh, linear, linear]	[tanh, tanh]
Hidden layer dropout rate	[0.0104]	[0.0334, 0.4162, 0.0409]	[1.6e-7, 0.0681, 0.058]	[1.7e-3, 0.22]
Number of training epochs	393	156	494	149
Batch size	32	42	365	632
Optimization function	rmsprop	rmsprop	adam	adam
Learning rate	2.1e-4	1.6e-4	1.4e-4	8e-3

Source: Gonzalez (2018)

8 CONCLUSION AND FUTURE RESEARCHES

This work sought to investigate the effectiveness of a hybrid approach based on Deep Learning and Evolutionary Algorithm as a time series forecasting method. In order to make clear the motivations of this project, characteristics of time series data have been presented and described from the conceptual and technical perspectives. Furthermore, it is put into evidence the challenges involved in developing time series forecasting models.

Deep learning networks are able to automatically learn arbitrary complex mappings from inputs to outputs and support multiple inputs and outputs. Therefore, these methods offer a lot of promise for time series forecasting, such as the automatic learning of temporal dependence and the automatic handling of temporal structures like trends and seasonality. Therefore, three different deep learning models have been selected and tested: Stacked Autoencoders (AE), Stacked Denoising Autoencoders (SDAE) and RNN with LSTM cells (LSTM). A standard Multilayer Perceptron Neural Network (MLP) is used as baseline since it is known as a classic time series forecasting method in the literature.

In the proposed methodology, the Covariance Matrix Adaptation Evolution Strategy (CMAES) algorithm is used for hyperparameter optimization, and thereby to improve the forecasting accuracy of each forecasting model. CMAES is powerful and expose very desired scalability properties, since it is computationally cheap and natively supports parallel evaluations. Moreover, an important aspect of the CMAES optimization process is to ensure that even a small population converges to the global optimum, and that individuals do not get trapped in suboptimal regions of the solution space. The population of individuals (each representing a set of hyperparameter values) is evolved in search of hyperparameters which yield the best forecasting performance of the DNN.

Being aimed at making it clearer the characteristics of the proposed approach, the results are compared on both artificial and real datasets. The artificial series are generated using the Mackey-Glass and Lorenz System equations. To investigate the performances of the proposed approach on real data, it is used the dataset obtained from the 2017 Global Energy Forecasting Competition (GEFCom 2017), which represents the hourly energy demand of the New England zone, US, provided by ISO New England. These datasets aim to reproduce practical time series problems as they explore seasonality components and chaotic behavior, allowing to test the methods under different scenarios.

Modeling time series faces many of the same challenges as modeling static data, such as coping with high-dimensional observations and nonlinear relationships between variables.

However, the problem with ignoring time and processing time series data as static input is that the importance of time is not captured and the structure present in the data is disregarded. When taking this approach, the context of the current input frame is lost and the only time-dependencies that are captured is within the input size. In order to capture long-term dependencies, the input size has to be increased, which can be impractical for multivariate signals or if the data has very long-term dependencies. For these reasons, even though many unsupervised feature learning models offer to relieve the user of having to come up with useful features for the current domain, there are still many challenges for applying them to time series data. The solution is to use a model that better incorporates temporal coherence, performs temporal pooling, or models sequences of hidden unit activations.

Through the experiments and the analysis of the results, it was found that the proposed approach can be properly trained and used in the short-term forecasting of time series with acceptable accuracy. In an overall analysis, all proposed deep learning methods presented relevant results and obtained significantly good performance on all studied cases. The RNN with LSTMs cells presented the best performance in both data sets with seasonal components (i.e., Mackey-Glass and hourly energy demand) and obtained relevant results on the Lorenz system data, proving to be the model that best learns the temporal dynamics of the data. MLP outperformed all other methods in the Lorenz system data set. As explained before, models that apply unsupervised pre-training are not able to capture very well the time dependency in the data. This can be better understood by analyzing the results obtained by the two models that incorporates pre-training layers (i.e., SAE and SDAE): they performed poorly both in the training and in the testing phases on all test cases. Comparing the results of different models on multiple time series data sets revealed that CMAES effectively traverses the solution space and delivers consistent and very high-quality results. The experimental results showed that augmenting minimal DNNs and optimizing them using CMAES can be an effective tool for challenging data sets. Therefore, it has been concluded that Deep Learning models combined with Evolutionary Algorithm are a promising alternative for time series forecasting problems.

8.1 Future Researches

Deep learning methods offer better performance on different time series problems compared to shallow approaches when configured and trained properly. There is still room for improving the learning algorithms specifically for time series data, e.g., performing signal

selection that deals with redundant signals and captures both short and long-term time dependencies. Further research in this area is needed to develop algorithms for time series modeling that learn even better features and are easier and faster to train. Another point to be investigated is the use of pre-training models that would be more focused on dynamic data such as Conditional RBM (Sutskever et al., 2009).

It is known that CMAES can be competitive especially in the regime of parallel evaluations. However, it is still needed to carry out a much broader and more detailed comparison, involving more test problems and comparing its performance against other optimization strategies. Moreover, as the deep learning architectures get increasingly complex and the number of hyperparameters grows, the consistency in the quality of the hyperparameters obtained by CMAES can be further analyzed by testing different parameters configuration, such as population size and learning rates. Therefore, future works can focus on developing a systematic methodology for balancing the exploration and exploitation of the hyperparameter space, and running CMAEs for larger DNNs.

REFERENCES

ABADI, M. et al. **TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems**. Available in: <https://www.tensorflow.org>. Visited on: September 2018.

AUGER, A.; HANSEN, N. A restart CMA evolution strategy with increasing population size. **In: Proc. of the 2005 IEEE Congress on Evolutionary Computation (CEC)**, vol. 2, pp. 1769-1776, 2005.

AZOFF, E. Michael. **Neural Network Time Series Forecasting of Financial Markets**. 1st. ed. New York, NY, USA: John Wiley & Sons, Inc., 1994.

BAHRAMI, S.; HOOSHMAND, R.; PARASTEGARI, M. Short term electric load forecasting by wavelet transform and grey model improved by PSO (particle swarm optimization) algorithm. **Energy**, v. 72, p. 434–442, 2014.

BARLOW, H. B. Unsupervised Learning. **Neural computation**, v. 1, n. 3, p. 295–311, 1989.

BENGIO, Y. Learning Deep Architectures for AI. **Foundations and Trends in Machine Learning**, v. 2, n. 1, p. 1–127, 2009.

BENGIO, Y. et al. Greedy layer-wise training of deep networks. **In: Proc. of the 19th International Conference on Neural Information Processing Systems 2006**, MIT Press, p. 153-160, 2006.

BENGIO, Y. Deep Learning of Representations: Looking Forward. **Lecture Notes in Computer Science**. p. 1–37. 2013.

BENGIO, Y.; SIMARD, P.; FRASCONI, P. Learning long-term dependencies with gradient descent is difficult. **IEEE Transactions on Neural Networks**, v. 5, n. 2, p. 157–166, 1994.

BERGSTRA, J. et al. Algorithms for Hyper-Parameter Optimization. **In: Proc of the 24th International Conference on Neural Information Processing Systems**, p. 2546–2554, 2011.

BOX, G. et al. **Time Series Analysis: Forecasting and Control**. John Wiley & Sons, 2015.

BRAMLETTE, M. Initialization, mutation and selection methods in genetic algorithms for function optimization. **In: Proc. of the 4th International Conference on Genetic Algorithms**, p. 100-107, 1991.

BREUEL, T. et al. High-Performance OCR for Printed English and Fraktur Using LSTM Networks. **In: Proc. of the 12th International Conference on Document Analysis and Recognition**, p. 683-687, 2013.

BUSSETI, E.; OSBAND, I.; WONG, S. Deep learning for time series modeling. **Technical report, Stanford University**, 2012.

CASDAGLI, M. Nonlinear prediction of chaotic time series. **Physica D. Nonlinear phenomena**, v. 35, n. 3, p. 335–356, 1989.

CHAKRABORTY, K. et al. Forecasting the behavior of multivariate time series using neural networks. **Neural networks: the official journal of the International Neural Network Society**, v. 5, n. 6, p. 961–970, 1992.

CHOLLET, F. **Keras: The Python Deep Learning library**. 2015. Available in: <https://keras.io>. Visited in: October 2018.

CIREŞAN, D. et al. Deep, big, simple neural nets for handwritten digit recognition. **Neural computation**, v. 22, n. 12, p. 3207–3220, 2010.

CLAESEN, M.; DE MOOR, B. Hyperparameter Search in Machine Learning. **arXiv preprint arXiv:1502.02127**, 2015.

COLLOBERT, R. et al. Natural Language Processing (Almost) from Scratch. **Journal of machine learning research: JMLR**, v. 12, n. Aug, p. 2493–2537, 2011.

DIETRICH, C. F. **Uncertainty, calibration and probability: the statistics of scientific and industrial measurement**. Routledge, 2017.

DIETTERICH, T. G. Machine Learning for Sequential Data: A Review. **Structural, Syntactic, and Statistical Pattern Recognition**, Springer, Berlin, Heidelberg, 2002.

DOYNE FARMER, J. Chaotic attractors of an infinite-dimensional dynamical system. **Physica D. Nonlinear phenomena**, v. 4, n. 3, p. 366–393, 1982.

EGGENSPERGER, K. et al. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. **NIPS Workshop on Bayesian Optimization in Theory and Practice**, Lake Tahoe, Nevada, USA, 2013.

ERTUGRUL, Ö. F. Forecasting electricity load by a novel recurrent extreme learning machines approach. **International Journal of Electrical Power & Energy Systems**, v. 78, p. 429–435, 2016.

FAMA, E. F. The Behavior of Stock-Market Prices. **The Journal of Business**, v. 38, n. 1, p. 34, 1965.

FAN, G. et al. Electric load forecasting by the SVR model with differential empirical mode decomposition and auto regression. **Neurocomputing**, v. 173, p. 958–970, 2016.

FAN, S.; CHEN, L. Short-term load forecasting based on an adaptive hybrid method. **IEEE Transactions on Power Systems**, [s. l.], v. 21, n. 1, p. 392–401, 2006.

FAN, Y. et al. TTS synthesis with bidirectional LSTM based recurrent neural networks. **In: Proc. of the of Interspeech**, pp. 1964–1968, 2014.

FILLIBEN, J. J. **Autocorrelation**. 2013. Available in: <https://www.itl.nist.gov/div898/handbook/eda/section3/eda35c.htm>. Visited on: September 2018.

FOX, R. A. **The Incorporated Statistician**, v. 11, n. 3, p. 170–171, 1961.

GHANBARI, A. et al. A Cooperative Ant Colony Optimization-Genetic Algorithm approach for construction of energy demand forecasting knowledge-based expert systems. **Knowledge-Based Systems**, v. 39, p. 194–206, 2013.

GOMEZ, F.; SCHMIDHUBER, J.; MIIKKULAINEN, R. Accelerated Neural Evolution through Cooperatively Coevolved Synapses. **Journal of machine learning research: JMLR**, v. 9, n. May, p. 937–965, 2008.

GONZALEZ, R. T.; BARONE, D. A. C.; PADILHA C. A. A. Ensemble system based on genetic algorithm for stock market forecasting. **In: Proc. of the 2015 IEEE Congress on Evolutionary Computation (CEC)**, 25-28 May, Sendai, Japan, p. 3102-3108. ISBN 978-1-4799-7491-7, 2015.

GRAVES, A.; MOHAMED, A.; HINTON, G. Speech recognition with deep recurrent neural networks. **2013 IEEE International Conference on Acoustics, Speech and Signal Processing**, p. 6645-6649, 2013.

GREENE, W. H. **Econometric analysis**. Pearson Education India, 2003.

GREFF, K. et al. LSTM: A Search Space Odyssey. **IEEE Transactions on Neural Networks and Learning Systems**, v. 28, n. 10, p. 2222-2232, 2015.

GROVER, A.; KAPOOR, A.; HORVITZ, E. A Deep Hybrid Model for Weather Forecasting. **In: Proc. of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining**, p. 379-386, New York, NY, USA, 2015.

HAMILTON, J. D. **Time series analysis**. Princeton university press Princeton, 1994.

HANSEN, N. **References to CMA-ES applications**. Available in: <http://www.cmap.polytechnique.fr/~nikolaus.hansen/cmaapplications.pdf>. Visited on: September 2018. 2009.

HANSEN, N. et al. A Method for Handling Uncertainty in Evolutionary Optimization with an Application to Feedback Control of Combustion. **IEEE Transactions on Evolutionary Computation**, v. 13, n. 1, p. 180–197, 2009.

HANSEN, N. The CMA Evolution Strategy: A Tutorial. **ArXiv e-prints, arXiv:1604.00772**, pp.1-39. 2016.

HANSEN, N.; BAUDIS, P.; AKIMOTO, Y. **cma package documentation**. Available in: <http://cma.gforge.inria.fr/apidocs-pycma/cma.html>. Visited on: September 2018.

HANSEN, N.; KERN, S. Evaluating the CMA Evolution Strategy on Multimodal Test Functions. **In: Parallel Problem Solving from Nature - PPSN VIII**, v. 3242, p 282-291, 2004.

HANSEN, N.; MÜLLER, S.; KOUMOUTSAKOS, P. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). **Evolutionary computation**, v. 11, n. 1, p. 1–18, 2003.

HANSEN, N.; OSTERMEIER, A. Completely derandomized self-adaptation in evolution strategies. **Evolutionary computation**, v. 9, n. 2, p. 159–195, 2001.

HAYKIN, S. **Neural Networks: A Comprehensive Foundation,** Prentice-Hall of India Pvt. Limited, 1999.

HEIDRICH-MEISNER, V.; IGEL, C. Neuroevolution strategies for episodic reinforcement learning. **Journal of algorithms & computational technology**, v. 64, n. 4, p. 152–168, 2009.

HERNÁNDEZ, L. et al. Artificial neural networks for short-term load forecasting in microgrids environment. **Energy**, v. 75, p. 252–264, 2014.

HINTON, G. E.; OSINDERO, S.; TEH, Y. A fast learning algorithm for deep belief nets. **Neural computation**, v. 18, n. 7, p. 1527–1554, 2006.

HIPPERT, H. S.; TAYLOR, J. W. An evaluation of Bayesian techniques for controlling model complexity and selecting inputs in a neural network for short-term load forecasting. **Neural networks: the official journal of the International Neural Network Society**, v. 23, n. 3, p. 386–395, 2010.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural computation**, v. 9, n. 8, p. 1735–1780, 1997.

HOLLAND, J. H. **Adaptation in Natural and Artificial Systems**. Ann Arbor, MI: University of Michigan Press, 1975.

HONG, T. **GEFCom2017**. Available in: http://www.drhongtao.com/gefcom/2017. Visited on: September 2018.

HONG, W. Electric load forecasting by seasonal recurrent SVR (support vector regression) with chaotic artificial bee colony algorithm. **Energy**, v. 36, n. 9, p. 5568–5578, 2011.

HONG, W. et al. Support Vector Regression with Chaotic Hybrid Algorithm in Cyclic Electric Load Forecasting. **In: Proc. of the International Conference on Soft Computing for Problem Solving,** Springer, India, p. 833–846 2012.

HÜSKEN, M.; STAGGE, P. Recurrent neural networks for time series classification. **Neurocomputing**, v. 50, p. 223–235, 2003.

HUSSAIN, A.; RAHMAN, M.; MEMON, J. Forecasting electricity consumption in Pakistan: the way forward. **Energy policy**, v. 90, p. 73–80, 2016.

HYNDMAN, R. J.; ATHANASOPOULOS, G. **Forecasting: principles and practice**. OTexts, 2018.

IGEL, C. Evolutionary Kernel Learning. **Encyclopedia of Machine Learning**. Boston, MA: Springer US, p. 369–373, 2010.

JAIN, A.; MAO, J.; MOHIUDDIN, K. Artificial neural networks: A tutorial. **IEEE Computer**, v. 29, n. 3, p. 31–44, 1996.

KENNEDY, J.; EBERHART, R. Particle swarm optimization. **In: Proc. of the IEEE International Conference on Neural Networks (ICNN)**, vol. 4, pp. 1942-1948, 1995.

KEOGH, E.; KASETTY, S. On the need for time series data mining benchmarks. **In: Proc. of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,** v. 7, n. 4, p. 349-371, 2002.

KLEIJNEN, Jack P. C. **Statistical Tools for Simulation Practitioners**. New York, NY, USA: Marcel Dekker, Inc., 1986.

KUREMOTO, T. et al. Time series forecasting using a deep belief network with restricted Boltzmann machines. **Neurocomputing**, v. 137, p. 47–56, 2014.

LEE GILES, C.; LAWRENCE, S.; TSOI, A. C. Noisy Time Series Prediction using Recurrent Neural Networks and Grammatical Inference. **Machine learning**, v. 44, n. 1-2, p. 161–183, 2001.

LÓPEZ-CARABALLO, C. H. et al. Mackey-Glass noisy chaotic time series prediction by a swarm-optimized neural network. **Journal of physics. Conference series**, v. 720, n. 1, p. 012002, 2016.

LORENZ, E. N. Deterministic Nonperiodic Flow. **Journal of the Atmospheric Sciences**, v. 20, n. 2, p. 130–141, 1963.

LORENZO, P. R. et al. Particle Swarm Optimization for Hyper-parameter Selection in Deep Neural Networks. **In: Proc. of the Genetic and Evolutionary Computation Conference Companion**, p. 1864-1871, New York, NY, USA, ACM, 2017.

LOSHCHILOV, I.; HUTTER, F. CMA-ES for Hyperparameter Optimization of Deep Neural Networks. **arXiv preprint arXiv:1604.07269**, 2016.

LOSHCHILOV, I.; SCHOENAUER, M.; SÈBAG, M. Bi-population CMA-ES Algorithms with Surrogate Models and Line Searches. **In: Proc. of the 15th Annual Conference Companion on Genetic and Evolutionary Computation**, p. 1177-1184, New York, NY, USA, ACM, 2013.

LÜTKEPOHL, H. **New Introduction to Multiple Time Series Analysis**. Springer Science & Business Media, 2005.

LV, Y. et al. Traffic Flow Prediction with Big Data: A Deep Learning Approach. **IEEE Transactions on Intelligent Transportation Systems**, v. 16, n. 2, p. 865–873, 2015.

MACKEY, M. C.; GLASS, L. Oscillation and chaos in physiological control systems. **Science,** v. 197, n. 4300, p. 287–289, 1977.

MARVUGLIA, A.; MESSINEO, A. Using Recurrent Artificial Neural Networks to Forecast Household Electricity Consumption. **Energy Procedia**, v. 14, p. 45–55, 2012.

MITCHELL, M. **An Introduction to Genetic Algorithms**. MIT Press, 1998.

MUKHERJEE, S.; OSUNA, E.; GIROSI, F. Nonlinear prediction of chaotic time series using support vector machines. **In: Proc. of the 1997 IEEE Signal Processing Society Workshop**, p. 511-520, 1997.

NALEPA, J.; KAWULOK, M. A Memetic Algorithm to Select Training Data for Support Vector Machines. **In: Proc. of the Annual Conference on Genetic and Evolutionary Computation**, p. 573-580, New York, NY, USA, ACM, 2014.

NELLI, F. An Introduction to Data Analysis. **Python Data Analytics: Data Analysis and Science Using Pandas, matplotlib, and the Python Programming Language**. Berkeley, CA, Apress, p. 1–12, 2015.

NG, A. et al. **Autoencoders and Sparsity**. 2013a. Available in: <http://deeplearning.stanford.edu/wiki/index.php/Autoencoders_and_Sparsity>. Visited on: April 2018.

NG, A. et al. **Multi-Layer Neural Network**. 2013b Available in: <http://ufldl.stanford.edu/tutorial/supervised/MultiLayerNeuralNetworks>. Visited in April 2018.

NG, A et al. **Stacked Autoencoders**. 2013c. Available in: <http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders>. Visited in: April 2018.

NIST/SEMATECH. **e-Handbook of Statistical Methods**. Available in: <https://www.itl.nist.gov/div898/handbook/>. Visited on: March 2018.

PASCANU, R.; MIKOLOV, T.; BENGIO, Y. On the difficulty of training recurrent neural networks. **In: Proc of the 30th International Conference on Machine Learning**, v. 28, p. III-1310-III-1318, 2018.

PRASAD, Sharat C.; PRASAD, Piyush. Deep Recurrent Neural Networks for Time Series Prediction. **Neural and Evolutionary Computing**, 2014

QIU, X. et al. Ensemble deep learning for regression and time series forecasting. **In: Proc. of the IEEE Symposium on Computational Intelligence in Ensemble Learning (CIEL)**, p. 1-6, 2014.

ROJAS, R. Genetic Algorithms. **Neural Networks: A Systematic Introduction**. Berlin, Heidelberg, Springer Berlin Heidelberg, p. 427–448, 1996.

ROMEU, P. et al. Time-Series Forecasting of Indoor Temperature Using Pre-Trained Deep Neural Networks. **In: Proc. of the 23rd International Conference on Artificial Neural Networks (ICANN)**, v.8131, p. 451-458, 2013.

ROSS, S. M. **Introduction to Probability Models**. Academic Press, 2014.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. **Nature**, v. 323, p. 533, 1986.

SCHMIDHUBER, J. Learning Complex, Extended Sequences Using the Principle of History Compression. **Neural computation,** v. 4, n. 2, p. 234–242, 1992.

SEIDE, F.; LI, G.; YU, D. Conversational speech transcription using context-dependent deep neural networks. **In: Proc. of the 12th Annual Conference of the International Speech Communication Association**, p. 437-440, 2011.

SNOEK, J.; LAROCHELLE, H.; ADAMS, R. P. Practical Bayesian Optimization of Machine Learning Algorithms. **Advances in Neural Information Processing Systems**, v. 25, p. 2951–2959, 2012.

SOUZA, J. C.; JORGE, H. M.; NEVES, L. P. Short-term load forecasting based on support vector regression and load profiling. **International Journal of Energy Research**, v. 38, n. 3, p. 350–362, 2014.

SUTSKEVER, I.; HINTON, G. E.; TAYLOR, G. W. The Recurrent Temporal Restricted Boltzmann Machine. **Advances in Neural Information Processing Systems**, v. 21, p. 1601–1608, 2009.

TAY, F. E. H.; CAO, L. J. Modified support vector machines in financial time series forecasting. **Neurocomputing**, v. 48, n. 1-4, p. 847–861, 2002.

TAYLOR, G. W. **Composable, Distributed-state Models for High-dimensional Time Series**. Phd - University of Toronto, 2010.

THIREOU, T.; RECZKO, M. Bidirectional Long Short-Term Memory Networks for predicting the subcellular localization of eukaryotic proteins. **IEEE/ACM Transactions on Computational Biology and Bioinformatics**, v. 4, n. 3, p. 441–446, 2007.

TOFALLIS, C. A better measure of relative prediction accuracy for model selection and model estimation. **The Journal of the Operational Research Society**, v. 66, n. 8, p. 1352–1362, 2015.

TSAI, J.; CHOU, J.; LIU, T. Tuning the structure and parameters of a neural network by using hybrid Taguchi-genetic algorithm. **IEEE Transactions on Neural Networks**, v. 17, n. 1, p. 69–80, 2006.

VIGNAUX, G. A.; MICHALEWICZ, Z. A genetic algorithm for the linear transportation problem. **IEEE Transactions on Systems, Man, and Cybernetics**, v. 21, n. 2, p. 445–452, 1991.

VINCENT, P. et al. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. **Journal of Machine Learning Research (JMLR)**, v. 11, n. Dec, p. 3371–3408, 2010.

VOJINOVIC, Z.; KECMAN, V.; SEIDEL, R. A data mining approach to financial time series modelling and forecasting. **International Journal of Intelligent Systems in Accounting, Finance & Management**, v. 10, n. 4, p. 225–239, 2001.

WEI, W. W. S. **Time Series Analysis: Univariate and Multivariate Methods**. Pearson Addison Wesley, 2006.

WHITLEY, L. Darrell; STARKWEATHER, T.; FUQUAY, D. Scheduling problems and traveling salesmen: The genetic edge recombination operator. **In: Proc. of the 3rd International Conference on Genetic Algorithms**, p. 133-140, 1989.

WILLIAMS, Christopher K. I.; RASMUSSEN, Carl Edward. **Gaussian processes for machine learning**. The MIT Press, v. 2, n. 3, p. 4, 2006.

WU, Jie et al. Short-term load forecasting technique based on the seasonal exponential adjustment method and the regression model. **Energy Conversion & Management**, v. 70, p. 1–9, 2013.

YALI, Nie. **A Multi-stage Convolution Machine with Scaling and Dilation for Human Pose Estimation.** 2018. 58 p. Thesis (Master degree Department of Electronic Engineering), Graduate School of Chonbuk National University, Republic of Korea, 2018.

YANG, Q.; WU, X. 10 Challenging Problems in Data Mining Research. **International Journal of Information Technology & Decision Making**, v. 5, n. 4, p. 597–604, 2006.

ZAZO, R. et al. Language Identification in Short Utterances Using Long Short-Term Memory (LSTM) Recurrent Neural Networks. **PloS One**, v. 11, n. 1, p. e0146917, 2016.

ZIVOT, E.; WANG, J. **Modeling Financial Time Series with S-PLUS**. Springer Science & Business Media, 2013.