U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Rui Gonçalves

Multivariate Time Series Analysis with Deep Learning

Doctoral Program in Computer Science
of the Universities of Minho, Aveiro and Porto

March 2021

Rui Gonçalves

Multivariate Time Series Analysis with Deep Learning

Thesis submitted to Faculty of Engineering of the University of Porto for the Doctor Degree in Computer Science within the Joint Doctoral Program in Computer Science of the Universities of Minho, Aveiro and Porto

Department of Electrical and Computer Engineering
Faculty of Engineering of the University of Porto
March 2021

Author's declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: .....

DATE: .....

Acknowledgments

I would like to thank my supervisors Fernando Lobo Pereira and Ana Paula Rocha for their guidance, fruitful discussions related to the topic of this dissertation, and mainly for their friendship.

This work is deeply related to JBet project activities. Therefore, I would like to thank my trader colleagues José Antonio Rocha and Alvaro Miguel. Also, to researchers Vitor Ribeiro, Paulo Sousa Dias, and José Pinto for the many discussions and good conviviality during these years.

Next, I wish to thank Jeff Heaton, the founder of the Encog neural network software framework [Heaton, 2015]. His well-maintained forums, excellent tutorial materials, and prompt responses to questions from curious minds all over the world greatly aided my research.

Finally, I would like to thank my family for all this excellent support in my participation in the MAP-i Doctoral Program in Computer Science.

The research activities described in this dissertation were supported by a Fundação para a Ciência e Tecnologia doctoral grant (Ref. SFRH/BD/70987/2010).

Abstract

This dissertation concerns the design of Deep Learning architectures to process time series to efficiently generate forecasts. A time series is a collection of observations made sequentially, typically measured at uniform time intervals. The special feature of time series is that the analysis must consider the time order since data points are usually not independent. Time series are examined in hopes of discovering a historical pattern that can be exploited in the computation of a forecast. Examples occur in various fields ranging from economics to engineering, and analyzing time series constitutes an essential part of Statistics. Deep Learning is part of a broader family of Machine Learning methods. It relies on Artificial Neural Networks and is the base framework on which the presented methodologies take shape.

This dissertation's main objectives are to propose innovative attention mechanisms superimposed on Recurrent Neural Network and bi-dimensional Convolutional Recurrent Neural Network layers. A new padding method applied directly in convolutional layers is also disclosed, suitable for multivariate time series processing.

The respective methodologies are thoroughly presented, as well as their execution analysis. All architectures shown follow standard supervised learning treatment that consists of training the models and then testing them in new data. Several datasets are used to evaluate their predictive performance. Databases taken from the University of California Irvine repository focused on Human Activity Recognition, Air Pollution - $PM_{2.5}$ Concentration, and Household Electric Power Consumption, as well as a private dataset concerning betting exchange markets, are considered as case studies. These datasets require extensive preprocessing and data analysis. Additionally, for the betting exchange market's case study, an end-to-end framework is described to perform automated feature engineering and market interactions using the developed models in production. The main results demonstrate the overall positive impact of the proposed methodologies.

Resumo

Uma série temporal é uma coleção de observações feitas sequencialmente, normalmente medidas em intervalos de tempo uniformes. A característica especial das séries temporais é de que a análise deve considerar a ordem do tempo, uma vez que os pontos de dados geralmente não são independentes. As séries temporais são examinadas na esperança de descobrir um padrão histórico que possa ser explorado na computação de uma previsão. Existem vários exemplos deste tipo de dados em vários campos que vão desde a Economia à Engenharia. Deste modo, análise de séries temporais constitui uma parte essencial da disciplina de Estatística. Aprendizagem profunda faz parte de uma família ampla de métodos de aprendizagem máquina. É baseado em Redes Neuronais Artificiais e é o quadro de trabalho base no qual as metodologias apresentadas tomam forma.

Os principais objetivos desta dissertação são propor mecanismos inovadores de atenção em redes neuronais sobrepostos a camadas recorrentes e camadas convolucionais recorrentes tal como ConvLSTM2D. Além disso, é divulgado um novo método de 'padding' em camadas convolucionais bidimensionais, apropriado para processamento de séries temporais multivariável.

As respetivas metodologias são apresentadas exaustivamente, bem como a sua análise de execução. Todas as arquiteturas mostradas seguem o tratamento padrão de aprendizagem supervisionada que consiste em treinar os modelos e, em seguida, testá-los em novos dados. Vários conjuntos de dados são usados para avaliar o desempenho preditivo. São considerados casos de estudo bases de dados retiradas do repositório da Universidade de California Irvine com foco em Reconhecimento de Atividade Humana, Poluição do Ar - $PM_{2.5}$ Concentração e Consumo de Energia Elétrica Doméstica, bem como um conjunto de dados privado sobre mercados de bolsa de apostas. Todos esses conjuntos de dados requerem extenso pré-processamento e análise de dados. Além disso, para o estudo de caso do mercado de bolsa de apostas, um quadro de trabalho completo é descrita para realizar engenharia de dados completamente automatizada assim como interações de mercado usando os modelos em produção. Os principais resultados demonstram o impacto geral positivo das metodologias propostas.

Contents

List of Tables

List of Figures

Listings

Chapter 1

Introduction

The increasing volume of data and the importance of data analytics in speed and heterogeneity constitute a clear trend that the available big data era is here to stay. It includes a new form of strategic behavior and business dialogue. Data is currently considered one of the most valuable intangible assets globally. Its further rapid increase is key to the transformation and growth of companies and the mitigation of digital aliteracy. The Machine Learning (ML) field enables a disruptive change for business organizations and individual agents due to the present power of predictive, classification, clustering analytics, and data generation/approximation. Correctly harnessing data can help to achieve better analytical-driven decisions. It allows boosting competitive advantages due to the faster reactions to core needs in addition to improving the performance of stakeholders on the provision of answers for complex questions. Moreover, the prospect of human error is completely eliminated. Automating tasks can significantly reduce the overall process cycle cost. Entities driven towards this journey need to understand how each aspect of their business can be optimized to satisfy new digital targets and further growth potential.

Applied ML is challenging because the designing of a perfect learning system for a given problem is intractable. There is no best training data or best algorithm for one problem, only the best that can be discovered. ML's application is best thought of as a search problem for the fittest mapping of inputs given the knowledge and resources available for a given project. ML, in its basis, is an approximation of unknown underlying mapping function from inputs to outputs. Hence, ML's conceptualization as a search helps to rationalize the spot checking of algorithms and understand what is happening when algorithms learn.

The goal of ML systems is to learn a generalized mapping between input and output data to make skillful predictions from new instances drawn from the domain where the output variable(s) is unknown. In a statistical perspective on ML, the problem is framed as the

learning of a perfect but unknown mapping function f given input data X and associated output data Y:

$$Y = f(X) \tag{1.1}$$

We have a sample set of X and Y and do the best to come up with a function $\hat{f}$ that approximates f, such that we can make predictions $\hat{Y}$ given new examples X in the future:

$$\hat{Y} = \hat{f}(X) \text{ where } \hat{f} \approx f \tag{1.2}$$

The learned mapping $\hat{f}$ will be imperfect. The form of the function f is unknown because, otherwise, we would not need a learning system and the solution would be specified directly. One needs to search $\hat{f}$ of the true underlying f that is good enough for the purpose. There are many sources of noise that introduce error into the learning process that can make it more challenging and, in turn, result in a less useful mapping. Also, there are many choices that a data scientist must make in order to optimize the process, such as:

- Framing of the learning problem;
- Observations used to train the system;
- How the training data is prepared;
- Form for the predictive model;
- Learning algorithm to fit the model on the training data;
- Performance measure by which to evaluate the predictive skill; etc.

There are many decision points in the development of a learning system, and none of the answers are known beforehand. All possible learning systems for a problem can be think of as a huge search in space, where each decision point narrows the search (see Figures 1.1 – 1.2). Also, there may be a natural hierarchy of this decisions, each of which further limits the search-space. This narrowing introduces a useful bias that intentionally selects one subset of possible learning systems over another with the goal of getting closer to a useful mapping. This biasing applies at the top level in the framing of the problem and at deeper levels configurations, such as the choice of learning algorithm and its parameterization.

On the top level, the chosen framing of the learning data used to train the system is a big point of leverage in the development of the learning system. In supervised learning, in which this dissertation is framed, there is normally some historical input-output pairs (X and Y) of data, which are used to train a predictive model. The chosen data to model the learning system must sufficiently capture the relationship between the X and Y for the training phase and production.

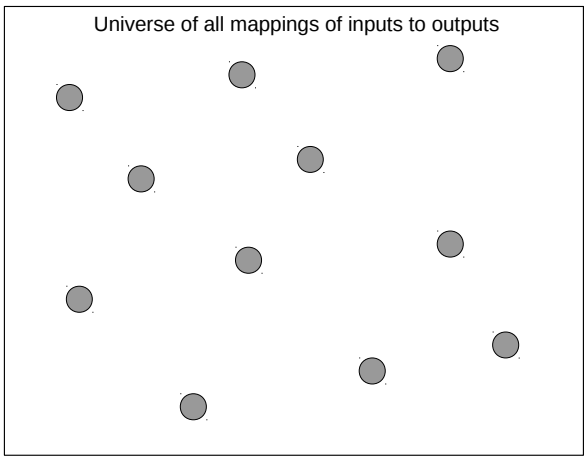


Figure 1.1: Some mapping functions $\hat{f}$ exploring the search space.

In the modeling phase, the representation of the model and the algorithm used to fit the model with the training data must be chosen. Again, this is another big point of leverage on the development of learning systems. Often, this decision is simplified to the selection of an algorithm, although it is common for stakeholders to impose conditions on the project (e.g, speed of execution), which create constraints on the form of the final model representation and on the scope of mappings in search. The algorithm used to learn the mapping will impose further constraints and, along with the chosen algorithm configuration, will control how the space of possible candidate mappings will navigate as the model learns (i.e., for ML algorithms that learn iteratively). Figure 1.2, shows that the act of learning from training

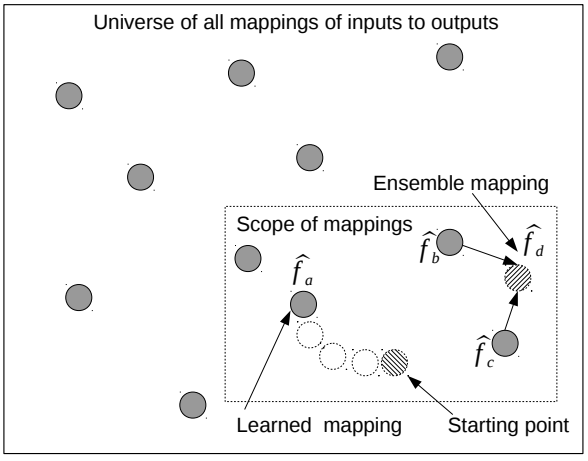


Figure 1.2: Constraints or selection of mapping functions ($\hat{f}$); Mapping function ($\hat{f}_a$) in the exploration process and a representation of mapping functions ($\hat{f}_b$ and $\hat{f}_c$) ensembled to produce a new mapping ($\hat{f}_d$).

data by a ML algorithm is, in fact, navigating the space of possible mappings for the learning system, hopefully moving from a poor mapping to a better mapping (e.g., hill climbing). This provides a conceptual rationale for the role of optimization algorithms at the heart of ML to get the best possible model representation for the specific training data.

We can also see that different models will occupy quite different locations in the space of all possible mapping functions. In turn, these have a quite different behavior when making predictions (i.e., uncorrelated prediction errors). This provides a conceptual rationale for the role of ensemble methods that combine predictions and/or architectures from different $\hat{f}_b$ and $\hat{f}_c$ to achieve a more skillful predictive model $\hat{f}_d$ (see Figure 1.2).

1.1 Deep Learning

There are many architectures to implement learning mapping functions. In this dissertation, we will focus on Deep Learning (DL) methodologies. Nevertheless, some other types of architectures will also be mentioned. DL is a sub-field of ML concerned with algorithms inspired by the structure and function of the brain called Artificial Neural Networks (ANNs).

Fast enough computers and enough data to actually train large Neural Networks (NNs) is at the core of DL. The main point of importance is scalability. As larger NNs are constructed and trained with additional data, their performance continues to increase. In general different from other ML techniques that reach a plateau in performance (Figure 1.3).

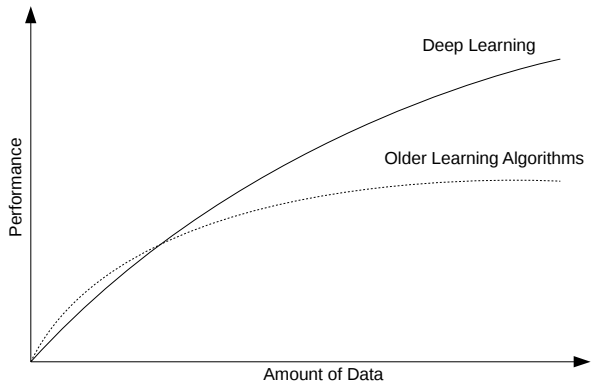


Figure 1.3: Data Science techniques scaling with amount of data, by Ng [2015].

In addition to scalability, another often cited benefit of DL models is their ability to perform automatic feature extraction from raw data, also called feature learning. Yoshua Bengio, one of the main experts in DL, began with a strong interest in the automatic feature learning that large NNs are capable of achieving. He describes DL in terms of the algorithmic ability to

discover and learn good representations using feature learning. Indeed Bengio [2012] claims: "*DL algorithms seek to exploit the unknown structure in the input distribution in order to discover good representations, often at multiple levels, with higher-level learned features defined in terms of lower-level features*". Automatically learning features at multiple levels of abstraction allows a system to learn complex functions mapping the input to the output directly from data. Also, citing Goodfellow et al. [2016]: *"The hierarchy of concepts allows the computer to learn complicated concepts by building them out of simpler ones. If we draw a graph showing how these concepts are built on top of each other, the graph is deep, with many layers. For this reason, we call this approach to Artificial Intelligence (AI): DL"*. Another important characteristic in a DL framework is the possibility of defining a NN in a pipeline of different modules, which are trainable or have been pre-trained.

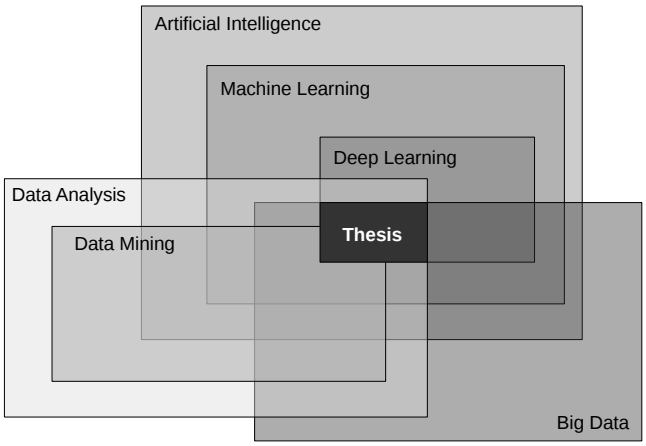


Figure 1.4: Locating this dissertation in the fields of AI, Data Analysis and Big Data.

In this dissertation, the main building blocks (i.e., type of layers) considered in the DL NN framework are:

- Fully connected perceptron layers, i.e., dense layers;
- Convolutional layers; and
- Recurrent layers, e.g., Long Short-Term Memory (LSTM) layers.

Because it has multiple stages in the process of recognizing a pattern, all of these stages are part of a deep modular training framework. DL methods represent multiple levels of information treatment, obtained by composing simple but non-linear modules that transform the representation of input information at one level. DL excels in problem domains where inputs (and even output) are analog representations which means that, besides few quantities of data in a tabular format, it can handle well pixel data images, text data documents or

audio data files. Figure 1.4 frames this dissertation within DL / Data Mining / Big Data fields for Multivariate Time Series (MTS) analysis.

1.2 Time Series Prediction Problem

Time series modeling is a dynamic area, which has attracted the attention of research community over the last few decades. The main aim of time series modeling is to carefully collect and rigorously study the past observations to develop an appropriate model to describe the inherent structure of data. This model is then used to generate future values of the time series. Time series forecasting can be termed as the act of predicting the future by understanding the past. Due to the indispensable importance of time series forecasting in numerous practical fields such as business, economics, finance, science, and engineering [Zhang, 2007, 2003, Tong, 1983], proper care should be taken to fit an adequate model to the underlying time series. Many efforts have been done by researchers for the development of efficient models to improve their performance. As a result, various important time series forecasting models have been improved.

A time series is a sequential set of data points, typically measured over successive fixed time intervals. It is mathematically defined as a vector x^t with $t \in [0, 1, 2, ..., T]$, where t represents the time elapsed and T is the length of the history data period for the time interval under analysis. Variable x^t is treated as a random variable, and, thus, the prediction $t + 1$ is stochastic. The measurements taken during an event in a time series are arranged in a proper chronological order. The goal is to forecast the variable of interest y^t at $t = T + m$ where m represents time steps ahead in the future. For this purpose, we need to define a model $\hat{f}$ with parameters θ for the variable of interest $\hat{y}^{T+m}$ where :

$$\hat{y}^{T+m} = \hat{f}(x, \theta) \tag{1.3}$$

A time series containing records of a single variable is termed as univariate. But if records have more than one variable (i.e., $n > 1$), it is termed as multivariate, where inputs of the model are given in the form of:

$$X_n^t = \begin{bmatrix} x_1^1 & \dots & x_1^t \\ & \vdots & \\ x_n^1 & \dots & x_n^t \end{bmatrix} \tag{1.4}$$

and the output prediction of a given variable is: $\hat{y}_n^{T+m} = \hat{f}(X, \theta)$. Figure 1.5 clarifies a matrix of inputs X_n^t.

A time series can be either continuous or discrete. In continuous time series observations are measured at every instance of time, whereas a discrete time series contains observations

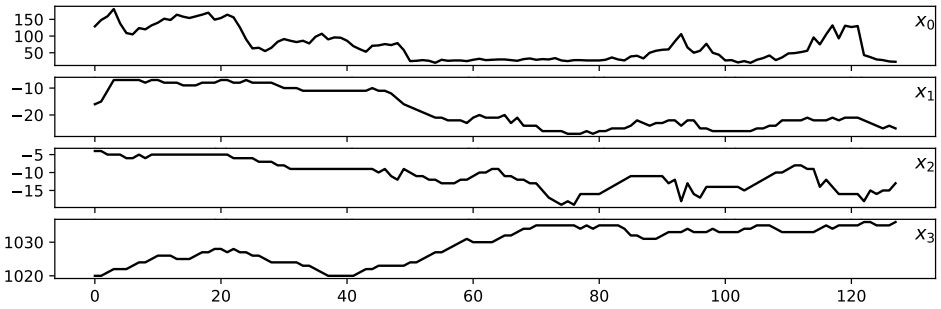


Figure 1.5: MTS input example. X_n^t with $n = 4$ variables containing $t = 128$ time steps for each x_n.

measured at discrete points of time. For example, temperature readings, flow of a river, and concentration of a chemical process can be recorded as continuous time series. On the other hand, population of a particular city, production of a company, exchange rates between two different currencies may represent discrete time series. As mentioned in Hipel and McLeod [1994], the variable being observed in discrete time series is assumed to be measured as a continuous variable using a scale of real numbers. Furthermore, a continuous time series can be transformed into a discrete one by merging data together over a specified time interval.

A time series is supposed to be affected by three main components, which can be separated from the observed data: trend, cyclical or seasonal, and irregular components. The general tendency of a time series to increase, decrease or stagnate over time is termed as secular trend or, simply, trend. Therefore, trend is a global long movement in a time series. In this dissertation, we will treat case studies where this component is minimal. The cyclical variation in a time series describes medium-term changes caused by circumstances, which repeat in cycles or seasons. Irregular or random variations in a time series are caused by unpredictable influences, which are not regular and also do not repeat in a particular pattern. These variations are caused by unpredictable incidences, external to the modeling problem. The excellent feature of ANNs, when applied to time series forecasting problems, is their inherent capability of non-linear modeling without any presumption about the statistical distribution characterizing the observations. The appropriate model is adaptively formed based on the given data. Due to this reason, ANNs are data-driven and self-adaptive by nature. During the past few years a substantial amount of research has been carried out towards the application of NNs for time series modeling and forecasting. In this dissertation, we aim to refine some techniques and propose novel DL based architectures.

A derivative concept from the described components of time series is the stationarity. It is a mathematical idea constructed to simplify the theoretical and practical development of stochastic processes [Box and Jenkins, 1990]. To design proper models, including DL-

based models, the underlying time series is expected to be stationary, which is not always the case. As stated by Hipel and McLeod [1994], the greater the time span of historical observations, the greater is the chance that the time series will exhibit non-stationary characteristics. However, for relatively short time span, one can reasonably model the series using a stationary stochastic process. Usually, time series showing trend or seasonal strong patterns are non-stationary in nature. In such cases, differencing and power transformations are often used to remove the trend and to make the series stationary (see Figure 1.6).

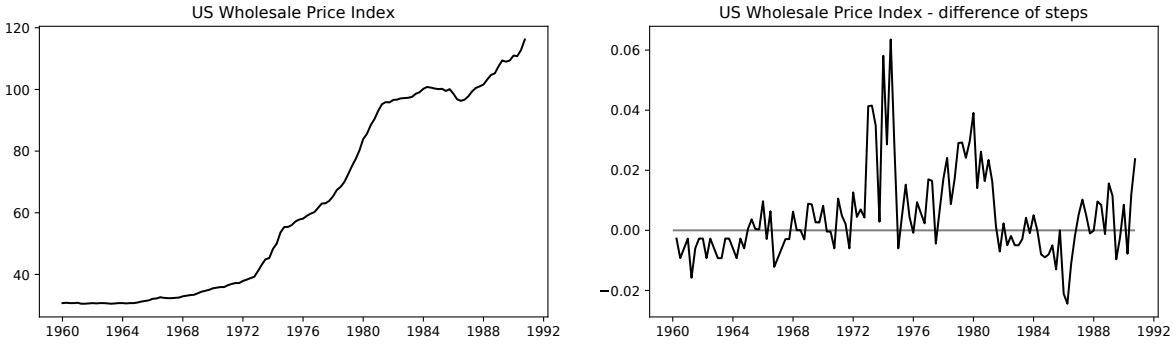


Figure 1.6: Time series example of US Wholesale Price Index (WPI). On the left subplot, non stationary trending raw data time series; On the right subplot, differencing it between time-steps making the time series stationary and more suitable to feed a model.

Regression and Classification in Time Series

Classification and regression are the two most common applications of NNs. Regression models predict a number, whereas classification tasks assign a non-numeric class to a given input. In a ML problem, first one needs to define whether we are dealing with a classification or regression type of problem, and get to know the analyzing target/output variable Y. Regardless whether the input X takes a continuous or discrete format, it is irrelevant to define the type of problem.

In a classification problem, one tries to predict a discrete number of values or labels. These generally come in categorical form and represent a finite number of classes. There are mainly two types of classifications that have influence on the type of model and configuration decisions:

1. Binary classification: when there are only two classes to predict, usually 1 or 0 values, e.g., fraud detection, anomaly detection; and

2. Multiclass classification: when there are more than two classes to predict, e.g., image classification problems where there can be than hundreds of classes.

An algorithm appropriate for binary classification in its basis, e.g., Support Vector Machines (SVMs), can be extended to multiclass classification using techniques like One-vs-All (OVA) and One-vs-One (OVO). OVA will train one classifier per class ending up with N classifiers. For class i, each classifier will assume i-labels as positive and the rest as negative. This often leads to imbalanced datasets meaning generic models might not work, but still there are some workarounds [Sridhar and Kalaivani, 2020]. In OVO technique, one trains a separate classifier for each different pair of labels. This leads to $\frac{N(N-1)}{2}$ classifiers. This is much less sensitive to the problem of imbalanced datasets, but it is more computationally expensive.

In the context of DL, NNs are suitable to both types of problems, binary and multiclass. However, the type of activation function for the output layer and error function in global, should be adequately selected (see Sections 2.1.1 and 2.2.4). Although in time series models,

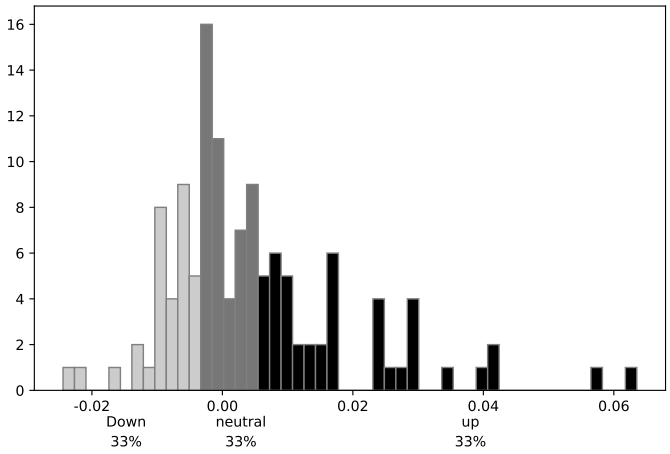


Figure 1.7: WPI differencing values (see Figure 1.6) frequency analysis to transform a regression into classification with egalitarian number of examples per class.

it is customary to predict the next number in line, i.e., work as a regression problem, in this dissertation we will be concern mainly with time series classification. For the case studies with an original regression format, this conversion is done by changing the target values using a frequency analysis technique. In doing so, we divide these values into classes such that a perfectly balanced number of examples per class is achieved for supervised learning. Figure 1.7 represents a histogram to establish the interval values, towards zero, for three classes in predicting WPI - going down, maintain, or going up - with 33% of samples in the dataset for each class. This way, we avoid an overfitting point of failure and set up the framework to extract results with more statistical significance and a more straightforward interpretation. For example, a problem with a high beam on the output histogram and a regression model that only predicts one exact value of that beam will have a low error in

theory. Still, in practice, it is biased and with low sensitivity. Moreover, hard to interpret. To see, in classification mode, how the model's response is spread over a spectrum of intervals can add interpretation value , e.g., confusion matrix analysis.

1.3 Objectives of the Research

This dissertation consists of three main objectives:

- *Design and investigate innovative convolutional attention mechanisms superimposed on simple recurrent and bi-dimensional convolutional recurrent based models.*
- *Design and investigate a new padding method designated by roll padding, which is applied to convolutional layers specifically projected for MTS processing.*
- *Combine the above new roll padding method with the bi-dimensional convolutional attention mechanism developed for bi-dimensional convolutional recurrent based models.*

Four families of models are adopted to enable the pursuit of the proposed research goals: Convolutional Neural Network (CNN), LSTM, Bi-dimensional Convolutional LSTM (ConvLSTM2D), and WaveNet based models. CNN-based models serve as a sandbox to demonstrate the potential of roll padding. LSTM-based models are used to compare the proposed attention mechanism with cases of no attention and standard attention. ConvLSTM2D-based models are used to verify whether the performance improves with the bi-dimensional attention mechanism proposed. We also compare the standard WaveNet with the developed WaveNet2D that uses bi-dimensional convolutions with roll padding. All methodologies are thoroughly presented, as well as their execution analysis. Several datasets are used to evaluate predictive performance of the models.

1.4 Approach

All architectures follow standard supervised learning scheme that consists of training the models with established input and output data, and, then use a segment of new data to compare the model's response with the real data. Databases taken from the University of California Irvine (UCI) repository focused on Human Activity Recognition (HAR), Air Pollution - Particulate Matter 2.5 ($PM_{2.5}$) concentration, and Household Electric Power Consumption, as well as a private dataset concerning betting exchange markets, are considered as case studies.

In terms of methodology, the following steps are taken. We start by data collection followed by Feature Engineering (FE) applied to inputs and outputs. All datasets require extensive

pre-processing and data analysis. Additionally, for the betting exchange market's case study, an end-to-end framework is fully described to perform Automated Feature Engineering (AFE) and market interactions using the models in production.

The modeling stage is the central point of this dissertation. Once the data pre-processing is done, the modeling stage begins where base architectures and respective add-ons are described. Afterwards, models are trained and tested in the learning phase. Once ready, models within each family are compared. This way, one can assess to which extent the proposed contributions improve the alternative conventional models in terms of predictive performance.

The remainder of this dissertation is organized as follows. Chapter 2 provides a literature review. Chapter 3 presents all case studies in detail and the applied FE. Chapter 4 presents methodologies and results. Conclusions and future work are compiled in Chapter 5.

1.5 Definitions of Terms

Automated Feature Engineering (AFE): The process where an algorithm automatically performs transformations to the input data to extract relevant information.

Adaptive Moment Estimation (Adam): A training update rule that automatically scales individual learning rates for each weight, taking into consideration past updates i.e. momentum. Deals well with sparse gradients, and can handle a stochastic objective function.

Autoencoder (AE): A NN that can learn efficient encodings for data to perform dimensionality reduction. Autoencoders typically have the same number of input and output neurons.

Bias term: The bias term performs a similar function as the y-intercept in linear regression.

Classifier: A NN, or another type of model, that is trained to classify its input into a predefined number of classes. A multiclass NN will typically use a softmax function on its output and have the same number of output neurons as the total number of classes.

Convolutional Neural Networks (CNN): Class of deep NNs that has successfully been applied to analyzing visual imagery. It is inspired by biological processes in that the connectivity pattern between neurons resembles the animal visual cortex's organization. It learns and uses several convolution kernels to process information.

Convolutional neuron: Matrix of weights to be learned. Also called in the literature by convolutional kernel, convolutional filter or feature detector. The notation used in this

dissertation for convolutional kernel is K.

Cross entropy loss function: A NN loss function, also simply called Log Loss (used in binary classification), that often provides superior training results for classification when compared to the quadratic loss function. For regression problems, Root Mean Square Error (RMSE) or Mean Square Error (MSE) should be considered.

Crossover: An evolutionary operator, inspired by the biological concept of sexual reproduction in which two genomes combine traits to produce offspring. Crossover performs the exploitation function of an evolutionary algorithm by creating new genomes based on fit parents.

Dataset: For supervised training, a dataset is a collection of input values (X) and the expected output values (Y). The presented research only deals with supervised training. The overall dataset is usually divided into training, validation, and test datasets.

Deep Learning (DL): A collection of training techniques and NN architecture innovations that make it possible to effectively train NNs with three or more layers.

Deep Neural Network (DNN): A NN with three or more layers.

Dot product based model : A model that uses dot products, or weighted sums, as a primary component of their calculation. NNs, linear regressions, and support vector machines for regression are all examples of dot product based models.

Engineered feature: A feature added to the feature vector of a model which is generated by applying a mathematical transformation to one or more features from the original feature vector.

Epoch: A unit in an iterative training algorithm where the entire training set has been processed.

Error function: Error, objective, cost, or loss function have the same meaning and are used to test the model performance. These terms can be used interchangeably, though some ML publications assign special meaning to them. In this dissertation, error function is the term mostly used.

Evolutionary algorithm: An optimization algorithm that evolves better-suited individuals from a population by applying mutation and crossover. The algorithm must balance between exploring new solutions and exploiting existing solutions to make them better.

Exploration: The process in a search algorithm where the search is broadened to new regions farther away from the best solution discovered so far.

Feature: Variables or attributes on the dataset, but sometimes also refferred to the output values of an intermediate layer, e.g., output of an encoder.

Feature Engeneering (FE): The process of creating new features by applying mathematical transformations to one or more of the original features.

Feature vector: The complete set of inputs to a NN layer. The feature vector must be the same size as the input of a layer.

Feed Forward Neural Network (FFNN): A NN that contains only forward connections between the layers.

Fully connected layer: Or dense layer, connect every neuron in one layer to every neuron in another layer. It is in principle the same as the traditional multi-layer perceptron NN.

Genome: An individual in an evolutionary program, candidate model solution. Complete set of one model parameters.

Gradient: A partial derivative of the loss function of a NN with respect to a single weight. The gradient is a key component in many NN training algorithms.

Hidden layer: A NN layer that occurs between the input and output layers. A NN can contain zero or more hidden layers.

Input layer: The first layer of neurons that receives the input to the NN.

Iteration: One unit of work in an iterative algorithm. If an iteration only processes a batch that is not the entire training set, then the iteration is called a step. A series of steps that process an entire dataset are called an epoch.

Layer: A collection of related neurons in a NN.

Learning rate: A numeric value that controls how quickly a model, such as a NN, learns. For backpropagation, the learning rate is multiplied by the gradient to determine the value to change the weight.

Linear regression: A simple model that computes its output as the weighted sum of the input plus an intercept value.

Local optima: A potential solution to an optimization problem that has no better solutions nearby in the search space. This local solution can prevent an optimization algorithm from finding other better solutions. These solutions are sometimes called local minima or local maxima, depending on if the optimization algorithm is seeking minimization or maximization.

Long Short-Term Memory (LSTM): A type of recurrent NN that uses a series of gates to learn patterns spanning much larger sequences than those that regular simple Recurrent Neural Networks (RNNs) are capable of learning.

Moment: Measure of the points in a probability density function. The zeroth moment is the total probability (i.e., one); the first moment is the mean; the second central moment is the variance; the third moment is the skewness; the fourth moment (with normalization and shift) is the kurtosis. The Adam training algorithm, used in this dissertation, estimates the first and second moments of the gradients.

Momentum: A numeric value that attempts to prevent a NN from falling into a local minimum. In its most basic form, this value is multiplied by the previous iteration's weight change to determine a value to add to the current iteration's weight change.

Mutation: An evolutionary operator, inspired by the biological concept of asexual reproduction, in which a single genome produces offspring with a slightly altered set of traits than the single parent. Because mutation introduces new stochastic information, it fulfills the exploration component of an evolutionary algorithm.

Nesterov momentum: A more advanced form of classic momentum that attempts to mitigate the effects of over correcting to a bad training batch.

Neural Network (NN): A mathematical model inspired by the human brain that is composed of an input layer, zero or more hidden layers, and an output layer.

One-Versus-All: A technique allows multiple binary classification models to perform multiclassification by training and combining several binary classification models. Each one classifies one class against the rest of the classes.

Output layer: The final layer of a NN. This layer produces the output. A regression NN will typically have a single output neuron. A binary classification NN will also have a single output neuron. A multiclass NN with three or more classes will have an output neuron for each class.

Principal Component Analysis (PCA): A form of dimension reduction that can shrink the size of an input vector to a smaller encoding with minimal loss of significance.

Recurrent Neural Network (RNN): A NN that contains backwards connections from layers to previous or same layers.

Regression: A NN or another model that is trained to produce a continuous value as its output. A regression NN will use a linear activation function on its output and have a single output neuron.

Regularization: Set of techniques that can prevent overfitting in NNs and thus improve the accuracy of a DL model when facing completely new data from the problem domain. They usually consist of reducing the NN complexity.

Root node: The node in a tree that is an ancestor of all other nodes. The root node for a single-node tree is also a terminal node.

Sample: also called example, and it is a record from a dataset.

Simple Recurrent NN (SRN): A NN with only a single recurrent connection, such as an Elman or Jordan network.

Softmax: An activation or transfer function that ensures that all outputs sum to 1.0, thereby allowing the outputs to be considered concurrent probabilities.

Stochastic Gradient Descent (SGD): An optimizer algorithm variant that uses a mini-batch that is randomly sampled each training iteration.

Test dataset: The portion of the data, outside of the training dataset, that test the model after the training phase. This dataset is sometimes referred to as out-of-sample data.

Time serie: Variable that evolves over time.

Training sataset: The data on which the model is trained. Usually, validation data are also kept so that the model can be evaluated on different data than it is trained with.

Transfer function: Also referred to as activation functions. They are applied to the neuron computation and defined by layers of a NN. All layers of a NN have transfer functions except the input layer.

Universal approximation theorem: A theorem that states that a FFNN with a single hidden layer containing a finite number of neurons can approximate continuous functions.

Update rule: The method by which a training algorithm updates the weights of a NN. Common update rules include: Nesterov accelerated gradient, Root Mean Square Propagation (RMSprop), and Adam.

Validation dataset: The portion of the data used to test and monitor the model during the training phase (between epochs). This data is outside of the training dataset.

Neural Network Basic Notation

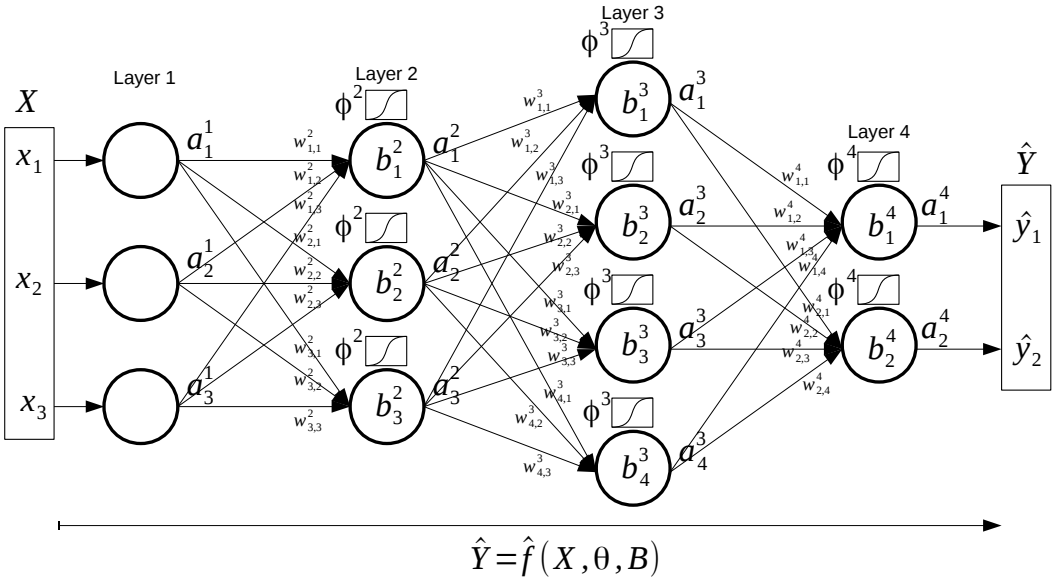


Figure 1.8: Notation used for deep FFNN models.

$\hat{f}()$	Mapping function representing a model.
X	Set of all inputs x_j. If the input relative to some layer l is multidimensional the notation $x^l_{ijk\ldots}$ may be used, with i, j, k representing the coordinates.
$\hat{Y}$	Set of outputs $\hat{y}_n$ or prediction of the model.
Y	Real target value(s) for a given example X.
θ	Set of all weights w^l_{jk}.
ϕ^l	Activation function used in layer l.
w^l_{jk}	Weight connection from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer.
a^l_j	Computation of the j^{th} neuron in the l^{th} layer : $a^l_j = \phi^l(\sum_k (w^l_{jk} \cdot a^{l-1}_k) + b^l_j)$, where a^1_j is the j^{th} element x_j in the input vector X.
z^l_j	a^l_j without using activation function : $z^l_j = \sum_k (w^l_{jk} \cdot a^{l-1}_k) + b^l_j$.
a^l	Compact version of a^l_j : $a^l = \phi^l(w^l \times a^{l-1} + b^l)$ for one layer computation.
a^L	Computation of all layers. Same as $\hat{f}$ computation.
b^l_j	Bias of the j^{th} neuron in the l^{th} layer.
B	Set of all b^l_j.
$J()$	Error/cost function, computes a difference or distance between $\hat{Y}$ and Y.
$\nabla J()$	Gradient of J, gradient is given by a vector whose components are the partial derivatives of J with respect to a particular weight: $\delta^l_{jk} \equiv \frac{\partial J}{\partial w^l_{jk}}$.

Summary

This chapter introduces the main relevant aspects for time series forecast. In particular, we provide the rationale for search space and mapping functions. DL framework is tackled since it is the basis of this dissertation. We define the scope of research and formalize the problem. Objectives and approach are also specified.

Chapter 2

Literature Review

The research conducted for this dissertation is focused on time series analysis. There is a considerable research community interest in this area. This Chapter reviews a representative sample of the current literature as it pertains to the dissertation research. The contents covered are: the deep learning framework, its building blocks and their evolution; the most prominent high-level architectures in the state-of-the-art with application in time series; NNs optimization methods; and finally, analysis of the classic auto-regressive method due to its relevance in MTS problems and also because it serves as a baseline comparison for some results of this dissertation.

2.1 Neural Networks

2.1.1 Feed Forward Neural Networks

This Section tackles the historical evolution of the FFNN base architecture. In this type of ANN, the information moves in one direction only, from input neurons to the output neurons, through hidden neurons if intermediate layers are present (see Figure 2.1). ANNs

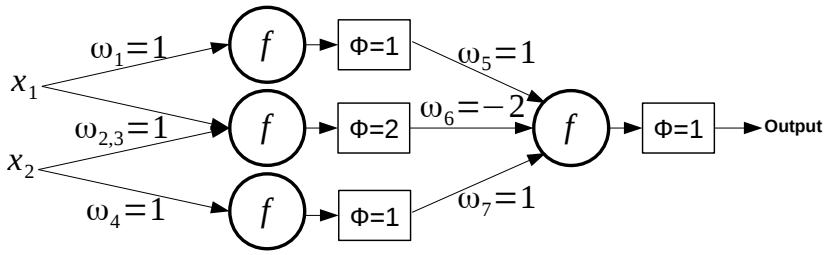


Figure 2.1: Multi Perceptron FFNN for Exclusive Or (XOR) operator.

are a processing paradigm inspired in the way the brain processes information. A class of algorithms composed by *MP-Units* (i.e., neurons emulation) was firstly introduced by Mcculloch and Pitts [1943]. Although NNs contain connections and neurons, they do not emulate every aspect of biological neurons. Modern NN are more of a mathematical model than a biological simulator. The seminal NN algorithm of McCulloch and Pitts specifies the calculation of a single neuron as the weighted sum of its inputs. The binary output $f(x) = [0, 1]$ is computed from the input as follows:

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x \geq \Phi \\ 0 & \text{else} \end{cases} \tag{2.1}$$

Where x is the input, w is the weight of the input and Φ the unit threshold.

Nearly all NNs created since their introduction are based upon feeding dot product calculations to activation functions over layers of neurons. DNNs simply have more layers of neurons. The research community has shown great interest in automating NN weights selection to achieve a particular objective. Hebb [1949] defined a process to describe how the connection strengths between biological neurons change as learning occurs. When the organism performs actions, connections increase between the neurons necessary for that action. This process became Hebb's rule, and it is often informally stated as "*neurons that fire together wire together*". Rosenblatt [1962] introduced the perceptron (see Figure 2.2)

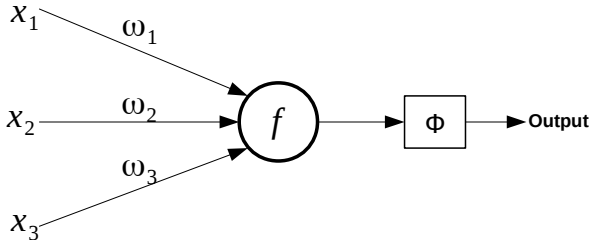


Figure 2.2: Perceptron Architecture.

that became the seminal work on NNs containing input and output layers. The perceptron is a two-layer NN with an input layer that contains weighted forward-only connections to an output layer. The transfer function defined for the classic perceptron keep a simple function that performs a threshold. Having N input values x_n, equation 2.2 shows the computation for each weigh w_n:

$$f(x) = \begin{cases} 1 & \text{if } \sum_{n=1}^{N} w_n \cdot x_n \geq \Phi \\ 0 & \text{if } \sum_{n=1}^{N} w_n \cdot x_n < \Phi \end{cases} \tag{2.2}$$

Rosenblatt [1962] demonstrated that a perceptron is incapable of learning the XOR operator, a non-linearly separable problem. Minsky and Papert [1969] also described severe limitations

in the use of a single perceptron in their seminal paper and monograph. There was a need to combine several perceptrons (see Figure 2.1) increasing the complexity in the optimal weights finding, i.e., training process. Nevertheless, disregarding computational restrictions, Cybenko [1989] formulated the universal approximation theorem and proved that a single hidden-layer NN (3 layers in total: input, hidden and output) with an arbitrary number os neurons, could approximate any function. The universal approximation theorem implies that additional hidden layers are unnecessary. However, while it is theoretically possible for a single-hidden layer NN, this outcome will not necessarily occur in practice. Hornik [1991] continued this research by also showing with a more pragmatic approach that a multilayer FFNN architecture gives NNs the potential to be universal approximators. Although FFNNs are universal approximators, they are not Turing [1936] complete without extensions that provide some type of memory state [Graves et al., 2014]. In other words, FFNNs can emulate any function, but do not e have the capacity to simulate computational arbitrary procedures.

The perceptron formulation was established, the output value of each neuron was given by equation 2.3 and it remains unchanged until the present date:

$$f(X, \theta, b) = \phi(\sum_{n=1}^{N}(w_n \cdot x_n) + b) \tag{2.3}$$

The parameter θ is used to represent the set of weights $\{w_1, ..., w_n\}$. X represents the list of inputs in the neuron containing $\{x_1, ..., x_n\}$. The function ϕ represents the transfer function or activation function, w_n is the weight on connection n, and x_n represents the input value n in the neuron. The variable b corresponds to the bias weight. Bias neurons enhance the NN's learning ability adding another degree of freedom [Cheng and Titterington, 1994]. Before the neurons at one NN layer can be processed, values of the neurons on the previous layers must be calculated. To generalize equation 2.3, for the computation a_j^l, i.e., any neuron in position j at layer l, we use the following notation:

$$a_j^l = \phi(\sum_k (w_{jk}^l \cdot a_k^{l-1}) + b_j^l) \tag{2.4}$$

where w_{jk}^l is the weight from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer, b_j^l is the bias on the j^{th} neuron in the l^{th} layer, and a_k^{l-1} is the previous layer computation on the connection coming from neuron k.

Activation Functions

An activation function is used to introduce non-linearity into a NN. This allows us to model a system that varies non-linearly with independent variables. Non-linear means the output cannot be replicated from a linear combination of inputs. This allows the model to

generate complex mappings from the available data. Hence, the NN becomes a universal approximator, whereas a model that uses a linear function (i.e., no activation function) is unable to make sense of complex data.

Another important aspect of the activation function is that it should be differentiable. This is required when we backpropagate through the network and compute gradients, and thus tune weights accordingly. These non-linear functions are continuous and transform the input into the range $[0, 1]$,$[-1, 1]$, etc. In a NN, it is possible for some neurons to have linear activation functions, but they must be accompanied by neurons with non-linear activation functions in some other part of the network. Although any non-linear function can be used as activation function, only a small fraction of these are used in practice. Listed below are some commonly used activation functions ϕ:

- **Binary Step**

$$\phi = f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{2.5}$$

A binary step function is generally used in the perceptron linear classifier (see Section 2.1.1).

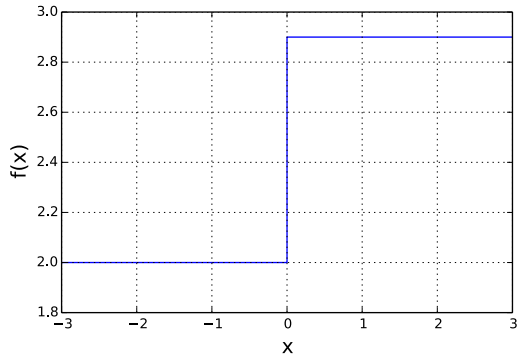


Figure 2.3: Binary step activation function.

It thresholds input values to 1 and 0, if they are greater or less than zero, respectively. This activation function is useful when the input pattern can only belong to one of two groups, i.e., binary classification (see Figure 2.3).

- **Sigmoid**

$$\phi = f(x) = \frac{1}{1 + \exp(-x)} \tag{2.6}$$

The sigmoid or *logistic* activation function maps input values in the range $[0, 1]$, which is essentially the probability of belonging to a class. So, it is mostly used for binary classification. However, it suffers from the vanishing gradient problem, as further explained bellow when the Rectified Linear Unit (ReLU) activation function is described. Also, the output produced is not zero-centered, which causes difficulties during optimization. It also causes a low convergence rate.

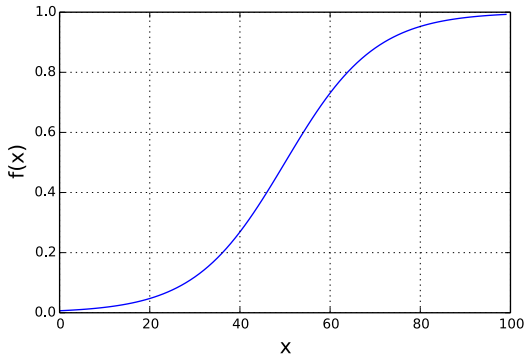


Figure 2.4: Sigmoid activation function.

- **Tanh**

$$\phi = f(x) = \tanh(x) \tag{2.7}$$

The tanh activaction function compresses the input into the range $[-1, 1]$ and it provides a zero-centered output. Therefore, large negative values are mapped to negative outputs and zero-valued inputs are mapped to near zero outputs. Also, gradients for tanh are steeper than sigmoid, but it also suffers from the vanishing gradient problem. tanh is commonly referred to as the scaled version of sigmoid. As exemplified in Figure 2.5, an alternative equation for the tanh activation function is given by:

$$\phi = f(x) = \frac{2}{1 + \exp(-2x)} - 1 \tag{2.8}$$

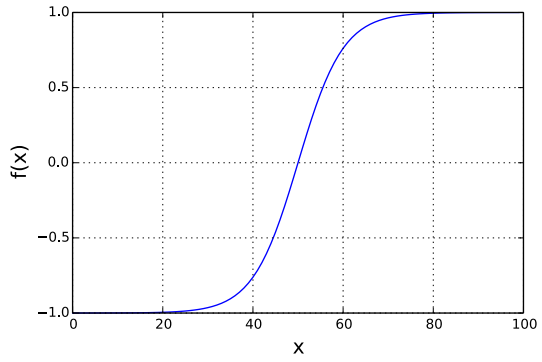


Figure 2.5: tanh activation function.

- **ArcTan**

$$\phi = f(x) = \tanh^{-1}(x) \tag{2.9}$$

The arctan activation function maps input values into the range $[-\pi/2, \pi/2]$. Its derivative converges quadratically against zero for large input values. In the sigmoid activation function, the derivative converges exponentially against zero, which can cause problems during backpropagation. Its graph is slightly flatter than tanh, so it has a better tendency to differentiate between similar input values.

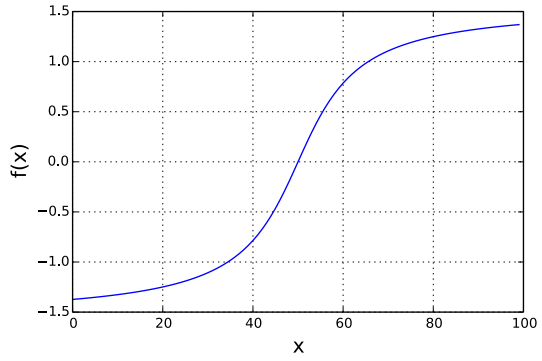


Figure 2.6: arctan activation function.

- **LeCun's Tanh**

$$\phi = f(x) = 1.7159 \tanh\left(\frac{2}{3}x\right) \tag{2.10}$$

This activation function was first introduced by LeCun et al. [1998b]. Constants in the above equation have been chosen to keep the variance of the output close to 1. LeCun et al. [1998b] states this activation function increases performance over sigmoid and unscaled tanh.

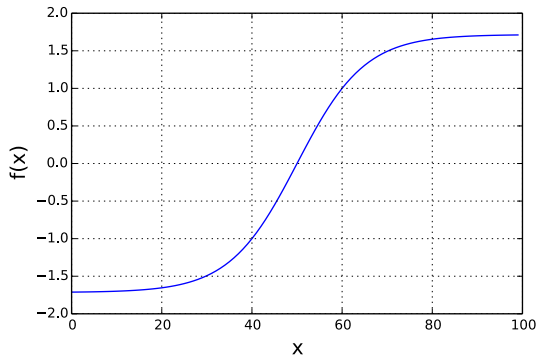


Figure 2.7: LeCun's tanh activation function.

- **Rectified Linear Unit**

$$\phi = f(x) = \max(0, x) \tag{2.11}$$

ReLU has the output 0 if its input is less than or equal to 0. Otherwise, its output is equal to the input (see Figure 2.8). It has been widely used in CNNs and it is also superior to the sigmoid and tanh activation functions, as it does not suffer from the vanishing gradient problem. Therefore, it allows for faster and effective training of DNN architectures.

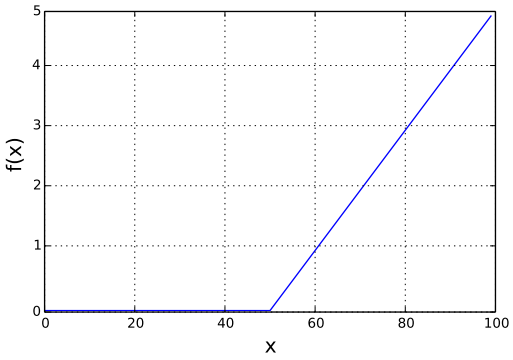


Figure 2.8: ReLU activation function.

Deepness in the NN causes the self-proclaimed vanishing gradient problem whenever using backpropagation in the training process. At each learning iteration, each NN weight receives

an update proportional to the gradient of the error function as described in Section 2.2.1. The vanishing gradient problem occurs when the error signal passes backwards and starts approaching zero. If the network is deep enough, the error signal from the output layer can be completely attenuated on the way back towards the input layer. Formally, attenuation exists because the derivative of the activation function ϕ' will always be near zero, especially for saturating neurons. Using the chain rule, which is backpropagation in NN terms, this almost zero derivative will multiply with the error signal before throwing it backwards at every level, i.e., layer. By iteratively throwing the error signal backwards, it becomes weaker and, hence, vanishing. Contrary to classical activation functions (e.g., arctan and sigmoid), the relatively new ReLU solves this concern [Arora et al., 2016]. ReLU is favorable not only due to the mitigation of the vanishing gradient problem but also because it forms highly sparse NNs, thereby inducing a more efficient and reliable performance given that the derivative takes constant values.

- **Smooth ReLU**

$$\phi_j^l = f(x_j^l) = \log\big(1 + \exp(x_j^l)\big) \tag{2.12}$$

Smooth ReLU, also known as the *softplus unit*, causes less saturation overall and overcomes the "*Dying ReLU*" problem by making itself differentiable everywhere. ReLU neurons output zero and have zero derivatives for all negative inputs (see Figure2.9). If the weights in the NN always lead to negative inputs into a ReLU neuron, that neuron is effectively not contributing to the NN training.

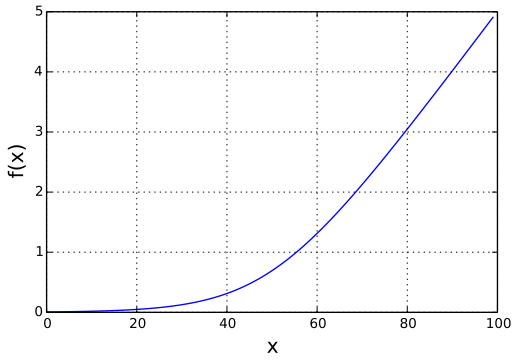


Figure 2.9: Smooth ReLU activation function.

- **Softmax**

$$\phi_j^l = f(x_j^l) = \frac{\exp(x_j^l)}{\sum_p \exp(x_p^l)} \tag{2.13}$$

The softmax function output tells the probability that classes where observations fall are true ones, thus it produces values within the range $[0, 1]$. It highlights the largest values and tries to suppress values which are below the maximum value (see Figure 2.10). Resulting values always sum up to 1. This function is widely used in classification logistic models and it is normally applied to the output layer only. Note that p is the number of classes (i.e., possible outcomes) to predict and distribute the resulting probabilities.

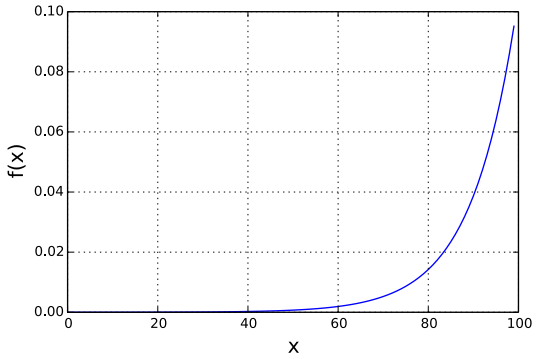


Figure 2.10: Softmax activation function.

2.1.2 Recurrent Neural Networks

By definition, a FFNN contains only forward connections. RNNs allow connections to previous or same layers. RNNs are characterized by connections that have loops, adding feedback and memory to the networks over time. Memory allows this type of NNs to learn and generalize across sequences of inputs rather than individual patterns. Elman [1990] and Jordan [1990] began the research on RNNs and introduced their SRNs as the *Elman and Jordan NN*. This SRN contains a context layer C, which memorizes the previous output computation of the recurrent neurons. Subsequent computations of the SRN will cause C to always loop-back the output produced by the recurrent layer in the next iteration. The information produced is stored and then used it the next iteration. The context layer in a SRN combines the new input with the information stored in the previous iteration (see Figure 2.11). For each time step shown to a SRN, an output is produced.

Knowing the size of the sequence that passes through the recurrent neuron, the chain of the context state updates can be flattened. This process is called unfold or unroll the

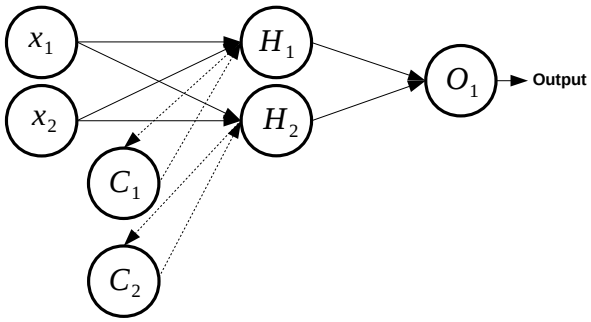


Figure 2.11: Elman SRN.

passage of information. The idea of RNN unrolling plays a relevant part in the way they are implemented for the backpropagation application. A technique known as backpropagation through time (BPTT) is normally used to train RNNs [Mozer, 1995, Robinson and Fallside, 1987, Werbos, 1988]. BPTT is the application of the Backpropagation training algorithm (see Section 2.2.1) to RNNs given sequences of data. Conceptually, BPTT works by unrolling all recurrent neurons a fixed number of times. Each input time step is processed and passed to the next state of the flatten recurrent neuron representation. The recurrent neuron output is produced at the final step of the sequence. Errors are then calculated and accumulated for each time step. The network is rolled back up, and the weights are updated. This way, BPTT can define a recurrent relation over time steps, being the context C state typically calculated using the following formula:

$$C^{t+1} = f(C^t * w_{rec} + x^t * w_n) \tag{2.14}$$

Where C^{t+1} is the recurrent neuron state at time step $t+1$, x^t an input of the sequence $x = \{x^1...x^t\}$ at time t. Parameter w_n is the weight in FFNN area and w_{rec} the weight in the recursive area of the RNN. Spatially, each time step of the unrolled RNN may be seen as an additional internal layer. Given the order dependence of the problem, the hidden state from the previous unrolled virtual layer is the input for the subsequent virtual layer, as illustrated in Figure 2.12. Moreover, h_t is the hidden output value produced at time step t. Note that h, as a vector, can also be outputted to the next layer of the global NN (e.g., using a *return_sequences* flag). This is normally used when staking recurrent layers.

The unroll technique is used to make the recurrent layer appear as one large FFNN. BPTT has a configuration parameter that specifies the number of time slices the program can unfold the RNN. Several virtual layers, equal to that configuration parameter, create the virtual network. The standard backpropagation algorithm used for FFNN trains the virtual network. FFNN without recurrent memory will always produce the same output for a given feature vector. However, RNN will maintain state from previous computations. This state will affect the RNN output. Therefore, the order that feature vectors are presented to the

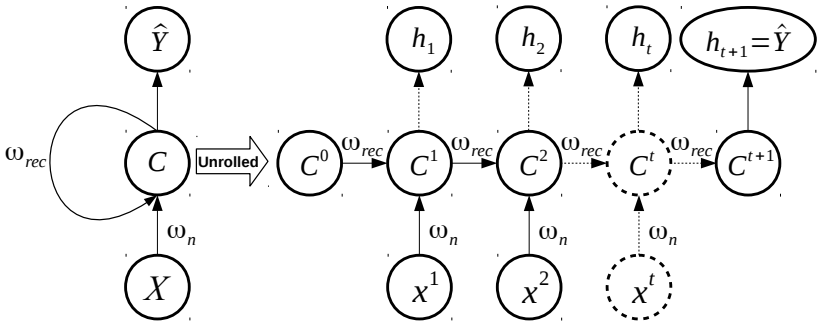


Figure 2.12: Unrolling a recurrent neuron

RNN will affect the output. Note that the RNN can be viewed as a state model with a feedback loop. The state evolves over time and feedback with a delay of one time step. This delayed loop gives the model memory since it can remember information between time steps. This capability makes RNNs suitable to time series prediction.

2.1.3 Long Short-Term Memory

One issue with standard RNNs, such as *Elman and Jordan NN*, is that the longer a time series becomes, the less relevant the context layer is. To overcome this problem, Hochreiter [1991] introduced the LSTM layer in his diploma thesis (see also [Hochreiter and Schmidhuber, 1997]). Hence, LSTM is a type of RNN capable of learning order dependence in sequence prediction problems and solving seamlessly problem settings composed by multiple input variables. This is a great benefit in time series forecasting since classical linear methods have difficulties to adapt the degree of time memory in MTS forecasting problems. Some authors are particularly clear and precise on articulating both the promise of LSTMs and how they work. LSTM are different from traditional FFNN. Bengio et al. [1994] claim that *"LSTMs... have an internal state that can represent context information. ... [they] keep information about past inputs for an amount of time that is not fixed a priori, but rather depends on its weights and on the input data... A RNN whose inputs are not fixed but rather constitute an input sequence can be used to transform an input sequence into an output sequence while taking into account contextual information in a flexible way."* The authors consider three basic requirements for LSTMs:

- The system should store information for an arbitrary duration;
- The system should be resistant to noise, i.e., fluctuations of the inputs that are random or irrelevant to predicting a correct output; and
- The system parameters should be trainable in reasonable time.

LSTMs also contain cycles that recursively feed the network activations from a previous time step as inputs to the network in order to influence predictions at the current time step. Furthermore, they use an extra memory cell when making predictions, but to this extent, this context memory cell required must also be learned. These activations are stored in the internal states of the network, which hold long-term temporal contextual information. This mechanism allows to exploit a dynamically changing environmental window over the input sequence history (e.g., Sak et al. [2014]).

The added cell state c to the basic RNN runs straight down the entire recursive chain, with only some minor linear interactions in order to control the information that needs to be remembered. LSTMs have the ability to remove or add information to the cell state by means of structures called gates. There are several variant architectures of LSTM units or hidden states. The common architecture for these units composes a memory cell, an input gate, an output gate and a forget gate:

- Memory cell stores a value (or state), for either long or short time periods. This is achieved by using an activation function for the memory cell;
- Input gate controls the extent to which a new value flows into the cell;
- Forget gate controls the extent to which a value remains in the cell; and
- Output gate controls the extent to which the value in the cell is used to compute the output activation of the LSTM unit.

Gates have in and out connections. The respective weights of the connections, which need to be learned during training, are used to orient the operation of the gates (see Figure 2.13). It is important to note that Figure 2.13 shows only a single step of the unrolled recurrent neuron/unit of a LSTM (i.e., the C recurrent cell in Figure 2.12, which is different from the c LSTM internal memory state). These LSTM neurons can be placed inside of regular FFNN. Usually, LSTM neurons are placed as an entire layer of such neurons. The c variable, in Figure 2.13, represents the context memory cell that is updated with $\tilde{c}$ value, depending on the i and f gates activation. The third gate o controls when the internal state is fed to the next recurrent step. The activate function used in the gates is σ sigmoid, while $\tilde{c}$ is obtained

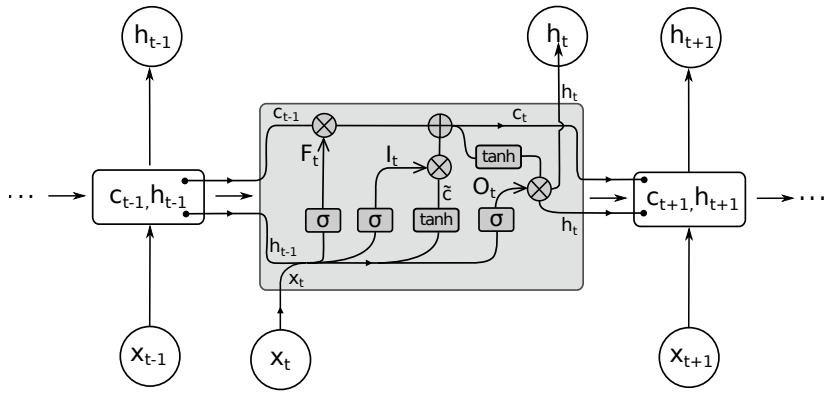


Figure 2.13: Illustration of the LSTM unit in the unrolled chain. i,f and o corresponds to the input, forget and output gates, respectively. $\tilde{c}$ and c_t denotes memory cell and memory cell update values, respectively.

using tanh activation function. All values are computed as follows:

$$f_t = \sigma(w_f[h_{t-1}, x_t] + b_f) \tag{2.15}$$

$$i_t = \sigma(w_i[h_{t-1}, x_t] + b_i) \tag{2.16}$$

$$\tilde{c} = tanh(w_{\tilde{c}}[h_{t-1}, x_t] + b_{\tilde{c}}) \tag{2.17}$$

$$c_t = f_t \times c_{t-1} + i_t \times \tilde{c} \tag{2.18}$$

$$o_t = w_o[h_{t-1,x_t}] + b_o \tag{2.19}$$

$$h_t = o_t \times tanh(c_t) \tag{2.20}$$

w_i w_f w_o and $w_{\tilde{c}}$ correspond to the weights on the input, forget, output gates and update candidate $\tilde{c}$ memory cell respectively. The same applies to bias term b. These gates are activated when the input reaches a threshold specified by the trained internal weights. The BPTT algorithm trains these gates threshold weights along with every other weight in the NN. Because the output hidden state in each recurrent computation is controlled by the input, forget, output gates and an extra memory, the LSTM is considerably more effective at recalling important parts of the time series sequence than a simple RNN. Unlike simple FFNNs without recurrence, LSTMs are specialized RNNs that can function as Turing-machines, i.e., they are Turing-complete [Pollack, 1990, Graves et al., 2014].

2.1.4 Convolutional Neural Networks

The study conducted by LeCun and Bengio [1994] is a seminal contribution to CNNs architectures. Traditionally, CNNs are applied to image classification [LeCun et al., 2004, Zeiler and Fergus, 2014]. These use an ad hoc architecture inspired by biological data

taken from physiological experiments done on the visual cortex. Lately, convolutional based methodologies have been also applied in temporal data [Halim and Kawamoto, 2020, de Oliveira e Lucas et al., 2020, Morid et al., 2020, Jiang et al., 2020, Fauvel et al., 2020, Hsu et al., 2020]. In this dissertation, we consider and adapt CNNs for MTS forecast problems given that the input can be seen as a bi-dimensional input likewise an image processing problem. In image recognition, CNNs expect and preserve the spatial relationship between pixels by learning internal feature representations. Instead of using correlations between pixels, we apply CNNs to learn patterns considering correlations between variables and their evolution in time. Note that convolutional layers can be applied to any dimensional type of input data [LeCun et al., 1995].

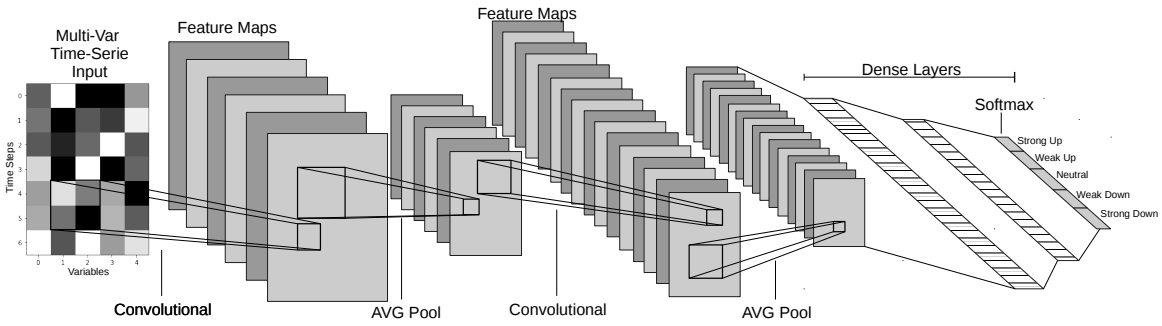


Figure 2.14: CNN 2D architecture for MTS forecast with classification

As clarified in Figure 2.14, there are normally three types of layers (i.e., building blocks) in a CNN : convolutional layers, pooling layers and fully connected layers (i.e., dense layers).

Convolutional layers are comprised of filters and feature maps. Filters are the weights of the layer that have to be learned. They are defined in matrices, and can have different dimensions according to the input dimension (e.g., 2D for gray images or 3D for rgb images). If the convolutional layer is an input layer, then the input will be one feature map, with the original values. If deeper in the network architecture, then the convolutional layer takes as input a set of feature map from the previous layer. One feature map is the output of one filter applied to the previous layer. In a convolutional layer, the number of "units or neurons" defined is the number of filters that generate that number of feature maps. A given filter is drawn across the entire previous layer, moved δ "pixels" at a time. The distance δ is also referred as stride. Each position results in an activation of the neuron and the output is collected in the new feature map. Lets consider a 4×4 input 2D matrix whose values are

only 0 and 1:

$$x^l = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \tag{2.21}$$

And another 2D matrix representing the values of a single convolutional neuron cell, i.e., filter K :

$$K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \tag{2.22}$$

Then, the Convolution of the 4×4 input with the 3×3 filter and stride $\delta = [1, 1]$, to generate one feature map, can be computed as shown:

$$\begin{bmatrix} 1_{\times 1} & 1_{\times 0} & 1_{\times 1} & 0 \\ 0_{\times 0} & 1_{\times 1} & 1_{\times 0} & 1 \\ 0_{\times 1} & 0_{\times 0} & 1_{\times 1} & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 4 & \dots \\ \dots & \dots \end{bmatrix}, \begin{bmatrix} 1 & 1_{\times 1} & 1_{\times 0} & 0_{\times 1} \\ 0 & 1_{\times 0} & 1_{\times 1} & 1_{\times 0} \\ 0 & 0_{\times 1} & 1_{\times 0} & 1_{\times 1} \\ 0 & 0 & 1 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 4 & 3 \\ \dots & \dots \end{bmatrix} \tag{2.23}$$

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 0_{\times 1} & 1_{\times 0} & 1_{\times 1} & 1 \\ 0_{\times 0} & 0_{\times 1} & 1_{\times 0} & 1 \\ 0_{\times 1} & 0_{\times 0} & 1_{\times 1} & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 4 & 3 \\ 2 & \dots \end{bmatrix}, \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1_{\times 1} & 1_{\times 0} & 1_{\times 1} \\ 0 & 0_{\times 0} & 1_{\times 1} & 1_{\times 0} \\ 0 & 0_{\times 1} & 1_{\times 0} & 1_{\times 1} \end{bmatrix} \Rightarrow \begin{bmatrix} 4 & 3 \\ 2 & 4 \end{bmatrix} \tag{2.24}$$

Pooling layers down-sample the previously generated feature maps. They follow a sequence of one or more convolutional layers and are intended to consolidate the features learned and expressed in the previous generated feature map. As such, they may be considered a technique to compress or generalize feature representations and generally reduce overfitting of the training data by the model. Pooling layers are often simple computations, taking the average or maximum of the input area in order to create mechanically (i.e., not subject to learning) its own feature map. Here is an example of a max pooling operation with kernel size 2×2 and stride $\delta = [2, 2]$

$$\begin{bmatrix} 1 & 1 & 2 & 4 \\ 5 & 6 & 7 & 8 \\ 3 & 2 & 1 & 0 \\ 1 & 2 & 3 & 4 \end{bmatrix} \Rightarrow \begin{bmatrix} 6 & 8 \\ 3 & 4 \end{bmatrix} \tag{2.25}$$

Many researchers disfavor the pooling operation and think that we can get away without it. For example, in *Striving for Simplicity* article, Springenberg et al. [2014] propose to discard the pooling layer in favor of architectures that only consist of repeated convolutional layers. To reduce the size of the representation, they suggest using larger stride in a convolutional

layer once in a while. Discarding pooling layers has also been found to be important in training good generative models, such as Variational Autoencoders (VAEs) [Hou et al., 2016] or Generative Adversarial Network (GAN) [Radford et al., 2015]. However, in addition to simple mechanic operations like max or average pooling, pooling layers can also perform many other types of functions. Gu et al. [2018] describes a variety of pooling types present in the literature.

Normally, for regression or classification problems, after passing through a sequence of convolutional and pooling layers, the feature maps are flatten (i.e., multidimensional information is flatted into one dimension) and fully connected layers are used to finalize the process. They are then used to create final non-linear combinations of features and for making the prediction outputs (see Figure 2.14).

2.1.5 Convolutional LSTM 2D

The ConvLSTM2D layer was proposed by Shi et al. [2015] to predict future rainfall intensity based on sequences of meteorological images. By using this layer in a DL NN model, authors were able to outperform state-of-the-art algorithms. The ConvLSTM2D is a recurrent layer similar to the LSTM, but internal matrix multiplications are exchanged with convolution operations. Since we are dealing with MTS problems, the concept of sequence segment is introduced to make inputs compatible with ConvLSTM2D layers, which in turn deal with segments of temporal sequences instead of sequence of images as highlighted in Figure 2.15.

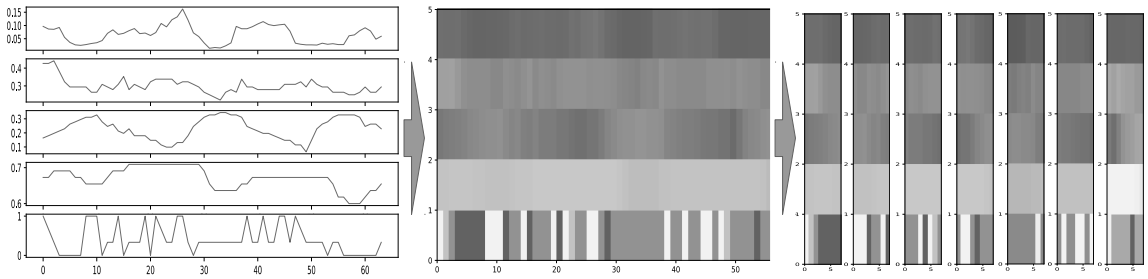


Figure 2.15: MTS input segmentation hack for ConvLSTM2D.

Data flow through the ConvLSTM2D cells by keeping a 3D format composed by *Segments* $\times$ *TimeSteps* $\times$ *Variables* rather than just a 2D input format composed by *TimeSteps* $\times$ *Variables* like in the standard LSTM. The ConvLSTM2D model can be useful for predictions in MTS problems that have constant and repetitive cycles that can be grouped and contained into a 2D map to be processed by 2D convolutions (e.g., Household Electric Power Consumption in Section 3.3 with cyclic information from day to day and over the week).

2.1.6 WaveNet

The WaveNet model is established in the mainstream since the contribution of van den Oord et al. [2016]. A key point of this model is the use of causal convolutions (i.e., adding zeros only on the left of the input map followed by the convolution operation), which ensures that the order of data modeling is not violated. As a result, the prediction $P(x_{t+1}|x_t, x_{t-1}, \dots)$ generated by the WaveNet model at time step $t + 1$ cannot depend on any future time steps. From a technical point of view, the causal convolution is implemented by shifting the input a few time steps behind. Since models with causal convolutions do not have recurrent connections, they are typically faster to train than RNNs. Nevertheless, one problem in causal convolutions is that they either require many layers or large filters to increase the receptive field.

In the left subplot of Figure 2.16, one can observe the use of causal padding in convolutions with kernel sizes of 4 and 3. Causal convolutions also allow to preserve the past time steps information, although transformed, for the next layer. This requires that the number of zeros to be added before the beginning of the sequence is given by kernel size $k - 1$.

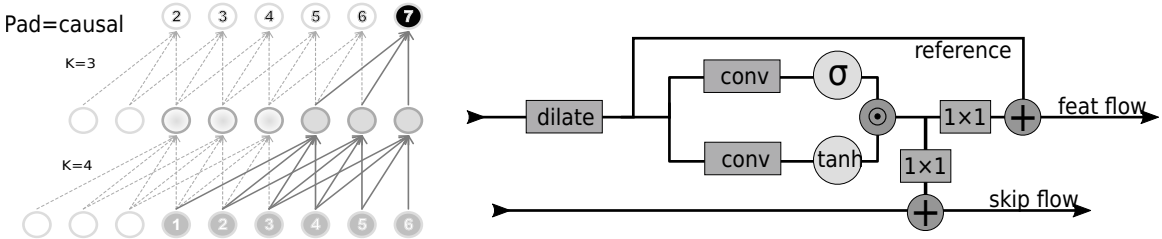


Figure 2.16: Causal padding on the left subplot and WaveNet residual block on the right subplot.

Another important characteristic of the WaveNet model is the use of dilated convolutions to gradually increase the receptive field. A dilated convolution is a convolution where the kernel K is applied over an area larger than its length k by skipping input values with a certain step. It is equivalent to a convolution with a larger filter derived from the original filter by dilating it with zeros. Therefore, when using a dilation rate dr (i.e., for $dr > 1$), the causal padding has size given by $dr \times (k - 1)$.

The residual block [He et al., 2015] is the heart of the WaveNet. As observed in the right subplot of Figure 2.16, it is constituted by two convolutional layers, one using sigmoid activation and another using tanh activation, which are multiplied. Then, inside the block, the result is pass through another convolution with $k = 1$ and $dr = 1$. This allows to downsample input channels and control the number of feature maps, whilst retaining the most important ones. Such technique is also heavily used in the InceptionNet architecture

proposed by Szegedy et al. [2014], often referred to as projection operation or channel pooling layer. Both residual and parametrized skip connections are used throughout the network to speed up convergence and enable training of much deeper models. This block is executed a given number of times in the depth of the network, with $N = \{1, ..., depth\}$. The dilatation dr increases exponentially according to the formula $dr = k^N$.

2.1.7 Temporal Convolutional Network

According to Bai et al. [2018], the Temporal Convolutional Network (TCN) model is characterized by three main characteristics:

(I) Convolutions in the architecture are causal, which means that there is no information moving from future to past since causal convolutions imply that an output at time t is convolved only with elements from time t and earlier (i.e., sequence modeling) from the previous layer;

(II) The architecture can take an input sequence of any length and map it to an output sequence of the same length similarly to RNNs. Similar to the WaveNet model, causal padding of length $k - 1$ is added to keep subsequent layers with the same length as previous ones; and

(III) It is possible to build long and effective history sizes using a combination of very deep networks augmented with residual blocks and dilated convolutions.

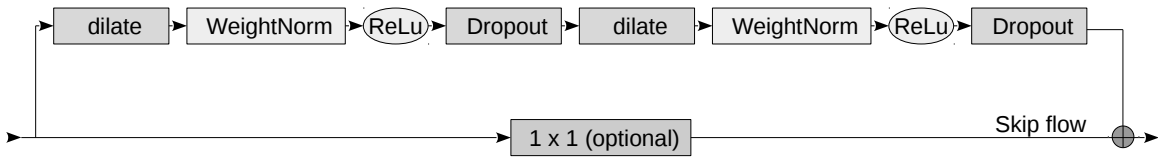


Figure 2.17: TCN residual block.

Therefore, a TCN model is derived from convolutional architectures for sequential data, designed to combine simplicity (i.e., no long skip connections across residual blocks like the WaveNet and gated activations compared to the LSTM) with autoregressive prediction.

As illustrated in Figure 2.17, the residual block of a TCN implies a series of transformations that effectively allow layers to learn modifications in the identity mapping at different depths rather than the entire transformation, which has been demonstrated to benefit very deep NNs [He et al., 2015]. Since the receptive field of a TCN model depends on the network depth N, kernel size k and dilation rate dr, the stabilization of a deeper and larger TCN constitutes a relevant task.

2.1.8 Autoencoders

AE is a NN used for unsupervised learning of efficient codings [Liou et al., 2014]. An AE aims to learn a representation – encoding – for a set of data, i.e., dimensionality reduction. AEs are typically used for data reconstruction, which can have a direct application in anomaly detection [Yin et al., 2020] and denoising information [Chen et al., 2020]. These type of models are normally trained to match the output with the input (or similar to the input like in denoising tasks). This type of training is considered unsupervised learning [Baldi, 2012].

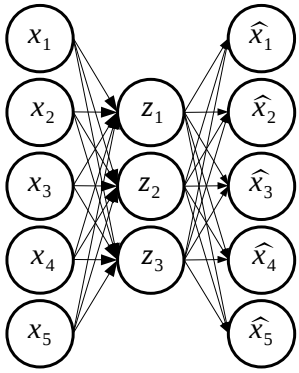


Figure 2.18: Simple AE of dense layers with one hidden layer

AE can also be used for normal classification or regression problems. In addition to the normal supervised learning, a first stage of unsupervised criterion can be relevant to help in classification or regression tasks. This two-stage technique is considered a semi-supervised learning approach, and is useful in scenarios where a large amount of unlabeled data along with small amount of labeled data is available. Exploiting the input part of the data, to encode it and regularize it, can serve as AFE technique, thus helping approaching better generalization error in the supervised phase [Erhan et al., 2010]. In this sense, an AE NN can be used to get a higher abstract level of information. The encoding part of the AE is used as a pre-trained model to feed and train a classification or regression NN with labeled data as illustrated in Figure 2.19.

To encode information, the AE is trained to make $\hat{x} = x$. Considering the MSE function (see Section 2.2.4) for the model represented in Figure 2.18, we have:

$$J(\theta) = \sum_{n=1}^{N} (x_n - \hat{x}_n)^2 \tag{2.26}$$

being θ the set of weights applied in the AE and N the number of input and output neurons. The output computation is made by $\hat{x}_n = a^2 a^1 x_n$. Moreover, a^l represents the computation of layer l, which in this case $l = 2$ is the output layer, i.e., decoding layer, and $l = 1$ is the

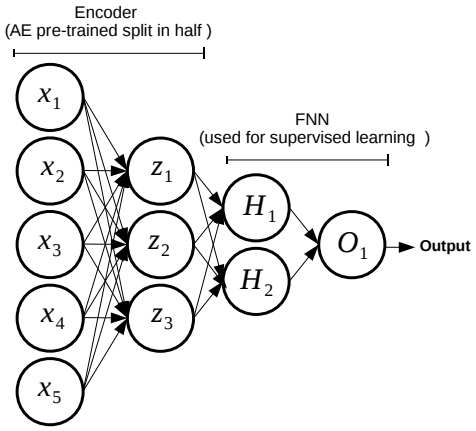


Figure 2.19: Pre-trained encoder used to feed a regression FFNN with generalized compressed information of the inputs.

middle layer, i.e., encoding layer. The error function becomes:

$$J(\theta) = \sum_{n=1}^{N} (x_n - a^2 a^1 x_n)^2 \tag{2.27}$$

This equation is similar to a PCA, which is a mathematical procedure that transforms a number of possibly correlated variables into a smaller number of uncorrelated variables called principal components. Nevertheless, AEs are much more flexible than PCA. NNs use have activation functions that introduces non-linearities to the encoding, whereas PCA can only represent linear transformations. Using the NN architecture representation also means that it is possible to stack AEs, called Stacked Autoencoders (SAEs) NNs. In addition, all other benefits of the DL framework can be applied such as backpropagation, regularization, dropout, different error functions, architecture flexibility, etc. SAEs can be constructed with dense layers as illustrated in Figure 2.18, recurrent, or even convolutional layers. When the input has temporal correlation, a LSTM SAE can be applied or, when there is spacial multidimensional correlation, CNN SAE can be used to compress information.

2.2 Optimization

2.2.1 Backpropagation

Backpropagation refers to two things:

- The mathematical method used to calculate derivatives and application of the derivative chain rule; and

- The training algorithm for updating network weights to minimize error.

The goal of backpropagation is to compute partial derivatives of the cost or error function $J(\theta_i)$ with respect to weight's w (being the set of all w's on iteration i : θ_i) in the NN. Currently, the backpropagation algorithm is the workhorse of learning in NN. The research for the automatic derivation of the weights of NNs started with the work of Werbos [1975] where the author mentions that the gradient descent could be used for the training of NNs. Previously, researchers only used gradient descent to find the minimum of functions. Rumelhart et al. [1985] were the first to apply gradient descent to NNs training. Backpropagation was thus introduced. Backpropagation was initially ineffective at training NNs with significantly more than two hidden layers and it was not known if NNs actually benefited from many layers [Bengio, 2009].

Backpropagation applies gradient descent to NN training. The gradient of each weight is the partial derivative of the loss function for that weight, while holding all other weights constant. The gradient of each weight is calculated and determines a change that should occur in the weight for the current training iteration. In backpropagation, the gradient of cost function $\nabla J(\theta)$, in conjunction with the NN computation applying the derivative of activation function, is used to compute the derivative error (i.e., *correction direction*) $\delta(\theta)$ using the set of weights $\theta = \{w_1, ...w_n\}$ via :

$$\delta(\theta) = \nabla J(\theta) \odot \phi'(\sum(\theta \cdot X) + b) \tag{2.28}$$

where $\phi'(\sum(\theta \cdot X) + b)$ is a concise notation representing the NN computation with the derivative of activation function (see equation 2.4).

Standard backpropagation computes a weight correction v for the iteration i in order to minimize the error function. v_i is calculated with the derivative error resulting from the previous weight $\delta(\theta_{i-1})$ scaled by the learning rate λ:

$$v_i = -\lambda \cdot \delta(\theta_{i-1}) \tag{2.29}$$

The weight correction v_i is applied to the previous NN weight's θ_{i-1} to obtain the new weight θ_i for the current iteration:

$$\theta_i = \theta_{i-1} + v_i \tag{2.30}$$

There have been several important enhancements to the basic backpropagation weight update rule. Momentum (γ) has been a significant component of backpropagation training for some time. Polyak [1964] introduced the seminal momentum algorithm that is a regularization technique for gradient ascent/descent. Momentum backpropagation adds a portion of the

previous iteration's weight change to the current iteration's weight change.

$$v_i = \gamma \cdot v_{i-1} - \lambda \cdot \delta(\theta_{i-1}) \tag{2.31}$$

Consequently, weight updates may have the necessary momentum to push through local minima and to continue the descent of the output of the loss function. Nesterov [1983] momentum was later applied to NNs, and further enhances the momentum calculation concept and increases the effectiveness with calculation of the gradient for current parameters plus the momentum constant multiplied by the previous calculated correction (v_{i-1}):

$$v_i = \gamma \cdot v_{i-1} - \lambda \cdot \delta(\theta_{i-1} + \gamma \cdot v_{i-1}) \tag{2.32}$$

Researchers have developed several innovations beyond the classic backpropagation and Nesterov momentum update rules. Classic backpropagation, even with Nesterov momentum requires that researchers choose learning rate and momentum training parameters that are applied across all weights in the NN. For these cases, it is usually advantageous to decay the learning rate as the NN trains [Bottou, 2012].

Silva and Almeida [1990] introduced the idea of a technique were each weight in a NN might benefit from a different learning rate. Duchi et al. [2011] took this idea further and developed the Adaptive Gradient algorithm (AdaGrad) to address both issues: decay the learning rate, as well as vary this rate per weight. Depending on the gradient force, the learning rate (i.e., step size for the descent on the error function) is adapted. Zeiler [2012] attempted to mitigate the aggressive monotonic learning rate decay of AdaGrad by defining a window of values for the gradients that affect the learning rate variation, named the Adadelta update rule .

Kingma and Ba [2014] introduced the Adam update rule that derives its name from the adaptive moment estimates that it uses. Adam estimates the first (mean) and second (variance) moments to determine the weight corrections. Adam begins with exponentially decaying averages of past gradients m:

$$m_i = \beta_1 \cdot m_{i-1} + (1 - \beta_1) \cdot \delta(\theta_i) \tag{2.33}$$

This average value is calculated automatically based on the current gradient $\delta(\theta_i)$. Kingma and Ba [2014] propose default values of 0.9 for parameter β_1. The update rule calculates the second moment v as follows:

$$v_i = \beta_2 \cdot v_{i-1} + (1 - \beta_2) \cdot \delta(\theta_i)^2 \tag{2.34}$$

Kingma and Ba [2014] propose the value of 0.999 for parameter β_2. The values m_i and v_i are estimates of the first moment (i.e., the mean) and the second moment (i.e., the uncentered

variance) of the gradients. They will have a strong bias towards zero in the initial training cycles. Because of it, the first moment's bias is corrected as follows:

$$\hat{m_i} = \frac{m_i}{1 - \beta_1} \tag{2.35}$$

Similarly, the second moment is also corrected:

$$\hat{v_i} = \frac{v_i}{1 - \beta_2} \tag{2.36}$$

These bias-corrected first and second moment estimates are applied to the ultimate Adam update rule as follows:

$$\theta_i = \theta_{i-1} - \frac{\lambda}{\sqrt{\hat{v_i}} + \epsilon} \cdot \hat{m_i} \tag{2.37}$$

The proposed value for ϵ parameter is 10^{-8}. This dissertation uses the Adam update rule as standard NNs training due to the rule's robustness to initial learning rate (λ) and other training parameters. Also, this option is justified due to the best results obtained during experiments.

NNs must start with random weights [Bengio, 2009]. These random weights are frequently sampled within a specific range. However, this simple range initialization can occasionally produce a set of weights that are difficult for backpropagation to train. As a result, researchers have shown interest for weight initialization algorithms that provide a good set of starting weights for backpropagation [Nguyen and Widrow, 1990]. Glorot and Bengio [2010] introduced what has become one of the most popular methods called the Xavier weight initialization algorithm. Because of its ability to produce consistently performing weights suitable for backpropagation training, the research experiments done in this dissertation uses the Xavier weight initialization.

2.2.2 Genetic Algorithms

Holland [1975] developed the concepts associated with Genetic Algorithms (GAs) in the book *Adaptation in Natural and Artificial Systems*. Another significant contributor in the field is Goldberg [1989, 2002]. GAs are adaptive search algorithms, which can be used for many fields, science, business, engineering, and medicine. GAs are adept at searching large and non-linear spaces. An extreme non-linear search space problem has a large number of local minimums and finding the optimal solution can be very difficult using conventional iterative gradient descent methodologies. GAs are most efficient and appropriate for situations in which:

- the search space is large, complex, or not easily understood; and

• there is no *hard coded* programming method that can be used to narrow the search space.

A GA possesses the ability to determine near optimal solutions in a reasonable time frame by simulating biological evolution. GAs closely resemble the biological model of chromosomes and genes. Individual organism in a GA generally consists of a single chromosome. The chromosomes are composed of genes. By manipulating the genes new chromosomes are created. These manipulations occur through crossover and mutation, just like they occur in nature. Crossover is analogous to the biological process of mating, and mutation is one way in which new information can be introduced into an existing population. In a GA each chromosome or individual represents one possible solution to the problem, and is composed by a collection of parameters to be optimized (i.e.,genes). Each chromosome or individual is initially assigned with a random collection of genes. This solution is used to calculate the fitness level, which determines the individual suitability to survive as in Darwin's theory of natural selection. Figure 2.20 illustrate how one NN individual solution can be represented in a chromosome.

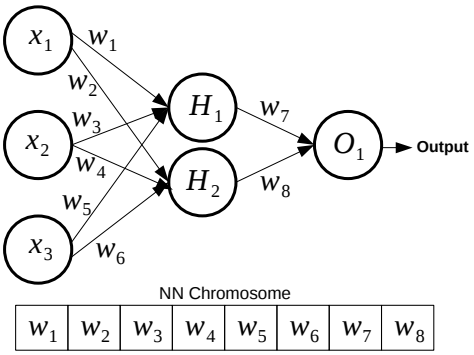


Figure 2.20: Transforming a NN into a linear chromosome θ_i for individual i.

The goal is to find the optimal weights for a NN solution. The GA works as follows:

1. Create an initial population of random possible solutions (i.e., chromosomes) spread across the search space;

2. Evaluate the fitness of each chromosome computing the associated NN, and using an error function;

3. Based on the fitness level of each chromosome, select the ones that will mate;

4. Crossover the selected chromosomes and produce offspring;

5. Randomly mutate some of the genes of the chromosomes in the population;

6. Evaluate the fitness of each individual and discard part (e.g., half) of the population with bad fitness; and

7. Repeat steps 3–6 until the best solution has not changed for a selected number of generations.

Crossover is achieved by selecting two parents and taking a *splice* from each other gene sequences. These *splices* normally divide the chromosome gene sequence into three parts. The children's are then created based on genes from each of these three sub-selections. This method can lead to a problem in which no new genetic material is introduced into the population. To introduce new genetic material, the process of mutation is used.

Mutation can be thought of natural experiments. These experiments introduce alterations in the genes of some individuals in the population. Natural selection will take care of the possibility of the mutation is good or bad for the individual not to stay alive or to mate. An important consideration for any GA is the mutation level that will be used. If the level of mutation is too high, the GA will be performing nothing more than a random search, and there will be no adaptation. If the mutation is too low, the GA will suffer from a too fast convergence problem (into a local minimum). Figure 2.21 illustrates the crossover and mutation process.

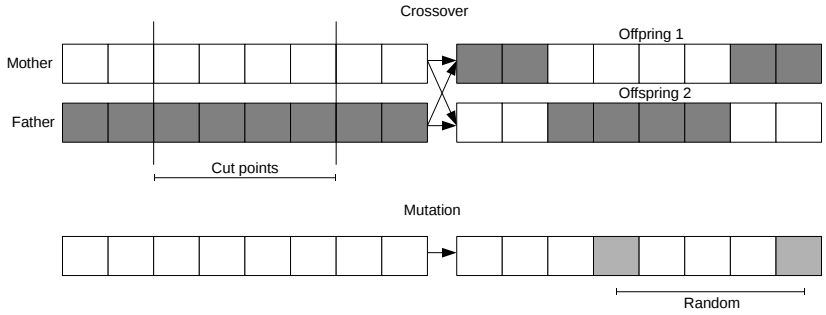


Figure 2.21: NN chromosome evolving with mating (crossover) and mutation.

GA has great potential to find, or stay close to, the global minimum of the error function due to the spread of the population (i.e., possible solutions) across the search space. As such, for problems with multiple local optimal points, with a noisy or stochastic (i.e., not smooth) objective function, GA can be a good choice. Also, in the context of DNNs, the vanishing gradient problem does not apply with GA training, so classical activation functions can be used, since the fitness score is obtained directly from the error functions without computing derivatives. The main disadvantage of this optimization method is that it is computationally expensive. However, it is again gradually gaining attention from the research community [Surakhi et al., 2020, Baldominos et al., 2020].

2.2.3 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is another metaheuristic search algorithm [Poli, 2008]. It is a promising optimizer since it was projected to avoid local minima on the error function. This algorithm is based on the migratory action of birds in a flock [Kennedy and Eberhart, 1995]. The basic idea is that, in a flock of birds looking for food, there is one bird located closest to the food. If that bird then communicates his proximity to the food with the other birds in the flock, they will swarm towards the area and search closely for food there. Likewise in GAs each bird is a candidate solution. The coordinates of the bird correspond to the variables to optimize. in the context of NNs, these dimensions are the weights θ. The search space has as many dimensions as the weights defined in the NN architecture.

Suppose θ_i^t denotes the position vector (i.e., weights values of one NN) of a given particle i in the multidimensional search space at iteration t. Then the position of each particle is updated in the search space as follows:

$$\theta_i^{t+1} = \theta_i^t + v_i^{t+1} \tag{2.38}$$

where v_i^t is the velocity vector (i.e., orientation) of particle i at iteration t that drives the optimization process and reflects both the own experience knowledge and the social experience knowledge from the all particles. Therefore, in a PSO method, all particles are initiated randomly and then evaluated to compute fitness together. The algorithm finds and saves the initial personal best for each particle and also the global best, i.e, best value of particle in the entire swarm. After that, a loop starts to find an optimal solution. In the loop, the velocity of particles is updated by personal and global bests, and then each particle's position is updated by the current velocity vector.

Each particle also has a personal best position in search space the is stored. The $P_{best,i}$ corresponds to the position in search space where particle i had the smallest value as determined by the error function $J(\theta_i^t)$. The $P_{best,i}^{t+1}$ for the next iteration is updated as follows:

$$P_{best,i}^{t+1} = \begin{cases} P_{best,i}^t & \text{if} J(\theta_i^t) > P_{best,i}^t \\ \theta_i^t & \text{if} J(\theta_i^t) < P_{best,i}^t \end{cases} \tag{2.39}$$

The global best G_{best} position at step t is calculated by:

$$G_{best}^t = \min\{P_{best,i}^t\}, \text{ where } i \in [1, ..., n] \text{ and } n > 1 \tag{2.40}$$

Note that G_{best} is the best position discovered by any of the particles in the entire swarm since the first iteration. Afterwards, the individual and social cognitive components can be

included in the velocity vector, whose calculation is given by:

$$v_i^{t+1} = v_i^t + c_1[P_{best,i}^t - \theta_i^t] + c_2[G_{best}^t - \theta_i^t] \tag{2.41}$$

where parameters c_1 and c_2 are positive acceleration constants used to level the contribution of the individual and social cognitive components respectively. The constant c_1 expresses how much confidence a particle has in itself, while c_2 expresses how much confidence a particle has in the group. Other parameters used in PSO are the maximum velocity of the particles and the number of particles used.

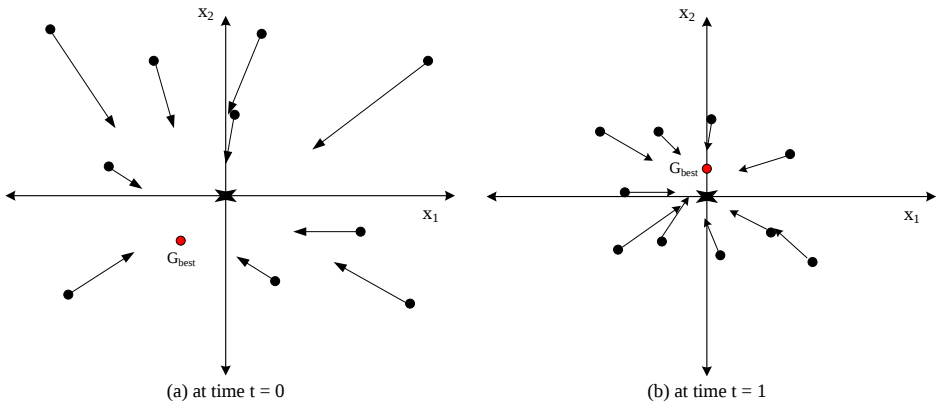


Figure 2.22: Particles moving in the search space using PSO, from one iteration to the next.

Figure 2.22 illustrate the convergence of n particles with position θ and update velocity vectors v in iteration t according to the new global best G_{best}^t found by the swarm. Since the search space in this illustration is bi-dimensional, the number of parameters optimized by particle is two (i.e., NNs with two weights).

2.2.4 Error Functions

The objective of a ML model, therefore, is to find parameters, weights or a structure that minimizes the error function $J(\theta, B, X^r, Y^r)$. It my also be referred to as loss or cost function. Since we are describing the error functions in detail, we will represent all arguments in this Section. They are the complete set of parameters on the NN: θ weights and B NN's set of bias terms; for a given example r in the training set with the inputs X and respective real output Y. NNs focus on the approach that illustrates statistical learning based on learning from data, thus minimizing an error function. Error functions are used to estimate how badly models are performing. It is a measure of how wrong the model is in terms of its ability to estimate the relationship between input X and Y. The error function gives a distance between the real value Y and the predicted value $\hat{Y}$. Depending on the type of problem, this

distance may be calculated in different ways to have the best statistical significance, thus influencing the quality of the generated model.

As explained in Section 2.2.1, backpropagation uses the gradient of error function $\nabla J(\theta, B, X^r, Y^r)$ (see equation 2.28) in order to use the update rule to direct the weights movement (v) towards a minimum, hopefully the global minimum. Thus, the respective gradient of the error function used in the computation is also provided. Next, are presented some of the most known and used error functions.

- **Quadratic cost**

Quadratic cost also known as MSE, maximum likelihood, and sum squared error, it is defined as:

$$J_{MST}(\theta, B, X^r, Y^r) = \frac{1}{n}\sum_{j}^{n}(a_j^L - Y_j^r)^2 \tag{2.42}$$

The gradient of this cost function with respect to the output of a NN and some sample r is:

$$\nabla J_{MST} = -\frac{2}{n}(a^L - Y^r) \tag{2.43}$$

where a^L corresponds to the complete computation of the NN (i.e., all layers processed: $\hat{Y} = a^L$). This error function is normally applied to regression problems since it outputs a comprehensive distance from optimal values Y to the predicted values $\hat{Y}$.

- **Root Mean Squared**

RMSE is very similar to MSE. Is given by the following equation:

$$J_{MST}(\theta, B, X^r, Y^r) = \sqrt{MSE} = \sqrt{\frac{1}{n}\sum_{j}^{n}(a_j^L - Y_j^r)^2} \tag{2.44}$$

and the derivative used for gradient calculation is given by:

$$\nabla J_{MST} = \frac{1}{2\sqrt{MSE}} \tag{2.45}$$

- **Cross-entropy**

Also known as log loss, the cross-entropy formula is given by:

$$J_{CE}(\theta, B, X^r, Y^r) = -\sum_{j}[Y_j^r \ln a_j^L + (1 - Y_j^r) \ln (1 - a_j^L)] \tag{2.46}$$

The gradient of this cost function relative to the output of a NN processing some sample r is:

$$\nabla J_{CE} = \frac{(a^L - Y^r)}{(1 - a^L)(a^L)} \tag{2.47}$$

This error function is normally applied to NNs with a softmax activation function on the output layer used for classification. It penalizes hard wrong classifications made with a high degree of certainty (i.e., wrong certainty).

- **Exponential**

The choice of this function requires to set a parameter τ. This is manually configured until the best results are achieved. The respective formula is given by:

$$J_{EXP}(\theta, B, X^r, Y^r) = \tau \ \exp(\frac{1}{\tau}\sum_j (a_j^L - Y_j^r)^2) \tag{2.48}$$

and the gradient is:

$$\nabla J_{EXP} = \frac{2}{\tau}(a^L - Y^r) J_{EXP}(\theta, B, X^r, Y^r) \tag{2.49}$$

- **Kullback-Leibler divergence**

This cost function is also known as information divergence, information gain, relative entropy, or KL divergence [Kullback and Leibler, 1951].

$$J_{KL}(\theta, B, X^r, Y^r) = \sum_j Y_j^r \log \frac{Y_j^r}{a_j^L} \tag{2.50}$$

The respective gradient used in computation for this function is given by:

$$\nabla J_{KL} = \frac{Y^r}{a^L} \tag{2.51}$$

2.3 Auto-Regressive Integrated Moving Average Model

Auto-Regressive Integrated Moving Average (ARIMA) models are a popular among economic research community and flexible class of forecasting models that utilize historical information to make predictions. The ARIMA model is included in this Section due to its relevance in MTS and also because it is used as a baseline comparison in regression analysis in Chapter 4. ARIMA is a general model for the time series which encapsulates the autoregressive model, non-seasonal differencing, and the moving average model [Box and Jenkins, 1990, Hipel and McLeod, 1994, Kirchgässner and Wolters, 2007].

Without integration, ARMA(p, q) model is a combination of AR(p) and MA(q) models and is suitable for univariate stationary time series modeling. In an AR(p) model the future value of a variable is assumed to be a linear combination of p past observations and a random error together with a constant term. Mathematically, the AR(p) model can be expressed by:

$$\hat{Y} = c + \sum_{i=1}^{p} \overset{ar}{w}_i x_{t-i} + \varepsilon_t = c + \overset{ar}{w}_1 x_{t-1} + \overset{ar}{w}_2 x_{t-2} + \cdots + \overset{ar}{w}_p x_{t-p} + \varepsilon_t \tag{2.52}$$

The model output $\hat{Y}$ corresponds to the x_t value of time series x at the present moment. It is the value we want to predict based on $t - i$, with $i = 1, 2, \ldots, p$. The integer constant p is known as the autoregressive order of the model. ε_t is the random error at time period t, $\overset{ar}{w}_i$ are parameters, and c is a constant term. Sometimes, the constant term c is omitted for simplicity. Usually, for estimating parameters of an AR process using a time series, Box and Jenkins [1990] equations are used.

Like AR(p) models regress against past values of the time series, MA(q) models use past errors as explanatory variables. The MA(q) model is given by:

$$\hat{Y} = \mu + \sum_{j=1}^{q} \overset{ma}{w}_j \varepsilon_{t-j} + \varepsilon_t = \mu + \overset{ma}{w}_1 \varepsilon_{t-1} + \overset{ma}{w}_2 \varepsilon_{t-2} + \cdots + \overset{ma}{w}_q \varepsilon_{t-q} + \varepsilon_t \tag{2.53}$$

where μ is the mean of the time series, $\overset{ma}{w}_j$ are parameters, with $j = 1, 2, ..., q$. Also, q is the moving average order of the model. Random shocks are assumed to be a white noise process, meaning a sequence of independent and identically distributed values (i.e., random variables with zero mean and a constant variance). Conceptually, MA model is a linear regression of the current observation of the time series against the random shocks of one or more prior observations. AR and MA models can be effectively combined together to form a general and useful class of time series models known as the ARMA models. Mathematically, ARMA(p, q) model is represented as:

$$\hat{Y} = c + \varepsilon_t + \sum_{i=1}^{p} \overset{ar}{w}_i x_{t-i} + \sum_{j=1}^{q} \overset{ma}{w}_j \varepsilon_{t-j} \tag{2.54}$$

Usually, ARMA models are manipulated using the lag operator notation. The lag or backshift operator is defined as $Ly_t = y_{t-1}$. Polynomials of lag operator or lag polynomials are used to represent ARMA models as follows:

$$\text{AR}(p) \text{ model: } \varepsilon_t = \overset{ar}{w}(L) y_t \tag{2.55}$$

$$\text{MA}(q) \text{ model: } y_t = \overset{ma}{w}(L) \varepsilon_t \tag{2.56}$$

$$\text{ARMA}(p, q) \text{ model: } \overset{ar}{w}(L) y_t = \overset{ma}{w}(L) \varepsilon_t \tag{2.57}$$

where $\overset{ar}{w}(L) = 1 - \sum_{i=1}^{p} \overset{ar}{w}_i L^i$ and $\overset{ma}{w}(L) = 1 + \sum_{j=1}^{q} \overset{ma}{w}_j L^j$.

ARMA models can only be used for stationary time series data. However, in practice, many time series contain trend and seasonal patterns and are non-stationary in nature. For this reason, Box and Jenkins [1990] proposed and Fischer and Planas [2000] refine the ARIMA model, which is a generalization of the ARMA model to include the case of non-stationarity. In ARIMA models, a non-stationary time series is made stationary by applying finite differencing to the data points. The mathematical formulation of the ARIMA(p, d, q) model using lag polynomials is given by:

$$\overset{ar}{w}(L)(1-L)^d y_t = \overset{ma}{w}(L)\varepsilon_t \text{ i.e. :} \tag{2.58}$$

$$\left(1 - \sum_{i=1}^{p} \overset{ar}{w}_i L^i\right)(1-L)^d y_t = \left(1 + \sum_{j=1}^{q} \overset{ma}{w}_j L^j\right)\varepsilon_t \tag{2.59}$$

Here, p, d and q are integers greater than or equal to zero and refer to the order of the autoregressive, integrated, and moving average parts of the model respectively. The integer d controls the level of differencing, i.e., the number of discrete differences the forecast variable's data has undergone in order to remove seasonality. Generally, $d = 1$ is enough in most cases. When $d = 0$, the ARIMA model is reduced to the ARMA(p, q) model.

To train the weights in ARIMA, it is normally used the combination of conditional sum of squares and maximization of the log-likelihood function, resulting in the CSS-Log-likelihood algorithm. More details and other related methods with a comparative study are reported by Safi and Saif [2014]. ARIMAX models extend ARIMA models through the inclusion of multiple exogenous variables X, meaning that multivariate inputs are captured [Pektas and Cigizoglu, 2013].

Sumary

In this Chapter, we discuss several model architectures and some types of modeling techniques to handle time series forecasts. Concerning the DL framework, the instantiation, parametrization, and inner workings of several components in NN models is described. The content is concentrated on the discussion of the main ones, those used as ground base for this dissertation. The following Chapters will discuss the exact methodologies that were used and developed to build MTS models. The next Chapter focuses on the case studies used to test the developed methodologies.

Chapter 3

Case Studies

In this Chapter, we describe the case studies used to mitigate the MTS modeling problem. Real world datasets were used to evaluate the developed DL methodologies.

The first case study is related to the HAR dataset from UCI. This is a well outlined problem and a very competitive dataset among the data scientist community [Shaohua Wan et al., 2020, Qin et al., 2020, Slim et al., 2019, Lockhart and Weiss, 2014, Anguita et al., 2013, Ronao and Cho, 2016, Sharma et al., 2008, Ignatov, 2018, Jiang and Yin, 2015, Romera-Paredes et al., 2013, Kaden et al., 2013]. By default, the HAR is a MTS classification problem. Data were extracted from smartphone accelerators and gyroscopic sensors in order to classify the type of action that is being performed by the user. Since this dissertation is also focus on MTS classification, minor FE procedures were applied to this case study. It also enables us to compare our results with other studies observed in the literature. This case study uses 2.56 seconds of information to predict the user activity. The sampling has 50% sliding window overlap.

The second case study provides a way for benchmarking the developed DL architectures against each other considering the air quality dataset of Beijing's $PM_{2.5}$ concentration [Sun et al., 2020, Pardo and Malpica, 2017, Li et al., 2017, Liang et al., 2015, Beijing US Embassy, 2014, Wang et al., 2013, Zhang et al., 2013]. The main goal is to predict the pollution level (i.e., $PM_{2.5}$ concentration) 12 hours ahead based on the last 72 hours information. The sampling methodology uses a 12 hours sliding window (i.e., 83% overlap).

The third case study is related to the individual household electric power consumption. This dataset is also provided by the UCI ML repository. In this dissertation, the target value is the average level of the global house active power consumption for the next 24 hours, categorized in five classes, based on the last 168 hours, i.e., 7 days. We use a sliding window of 24 hours (i.e., 86% overlap between examples). Normally, the metric used in the literature for this

dataset, to evaluate the performance, is RMSE [Koschwitz et al., 2018, Georges Hebrail, 2010, Chujai et al., 2013, Kim and Cho, 2018, Behera et al., 2016]. We focus on MTS classification, and so we provide a comparison of results between different DL methodologies using accuracy and categorical cross-entropy metrics.

The last case study is related to exchange markets. For this case study, we describe the software framework developed to collect data, FE, and put the models into production using trading strategies, i.e., simulate the AI agent actions in the market. This was an end-to-end ML project concerned with preparing complex data, training models on it, and then deploying those models. Indicator analysis and modeling is only a part of the full trading system. When building a forecast system that uses models and trading strategies, a global system evaluation to measure the performance of the combined components must also be done [Gonçalves et al., 2013, Goncalves et al., 2019]. This case study works over the last 10 minutes of pre-live betting exchange horse racing markets. 8 minutes are used to classify the price movement in the final 2 minutes before the beginning of the race, and there is no sliding window overlap, i.e., each race is one example.

3.1 Human Activity Recognition

The HAR dataset from the UCI ML repository is one of the datasets used to train, test and compare the methodologies described in this dissertation. This is a well-known and competitive dataset retried from smartphone sensors, which contains 3-axial gravitational acceleration, 3-axial body acceleration and 3-axial body gyroscope readings captured at a constant rate of 50Hz, totalizing 9 variables over 128 time steps. Readings were taken from 30 volunteers holding a smartphone to record six different types of activities: walking, walking upstairs, walking downstairs, sitting, standing, and laying. Overall, the UCI HAR dataset consists of 10299 examples. Figure 3.1 is the plot the 9 inputs for one train example of walking class output.

It is important to highlight that the UCI repository provides the separation of data into train and test datasets. The training set contains 7352 examples, while the testing set has 2947 examples. This working setup is referred to as 21-9, which means that 21 subjects are used for training and 9 subjects are used for testing. Models of this type of working setup are said to fall into the category of impersonal models Lockhart and Weiss [2014]. As clarified in Table 3.1, another relevant point is the number of examples *per* class. Any unbiased comparison of test results between different studies requires a persistent consistency on the adoption of these values.

In addition to the Raw Time Distributed (RTD) dataset, UCI also provides a FE dataset

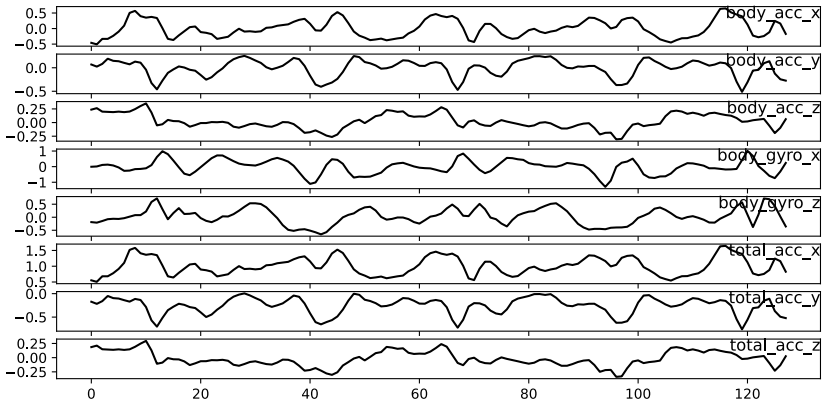


Figure 3.1: A sample example of HAR dataset for *walking* class.

	walking	upstairs	downstairs	sitting	standing	laying
Test set	496	471	420	491	532	537
Train set	1226	1073	986	1286	1374	1407

Table 3.1: Number of examples per class in the train and test HAR dataset provided by UCI.

that transforms the RTD data into 561 non-temporal features (e.g., average, max, min, etc.). Since the FE dataset maintains the order and number of examples equal to the RTD dataset, the performance of models that use the FE dataset can be compared to the performance of models that use the original RTD dataset.

Table 3.2 summarizes test results of some prominent studies. Romera-Paredes et al. [2013] proposes a OVO multi-classification SVM with a linear kernel for the classification task. The method uses majority voting to find the most likely activity for each test sample from an arrangement of 15 binary classifiers. Anguita et al. [2013] introduces the HAR dataset and obtain results exploiting a multi-classification SVM. Kaden et al. [2013] employs a sparse kernelized matrix Learning Vector Quantization (LVQ) model. Their method is a variant of LVQ in which a metric adaptation with only one prototype vector for each class is defined. Ronao and Cho [2016] presents a temporal Fast Fourier Transform (FFT) plus a CNN model. The temporal FFT concept was developed by Sharma et al. [2008]. It is used to process information that subsequently feeds a CNN. Similarly, Jiang and Yin [2015] applies a bi-dimensional Discrete Fourier Transform (DFT) to the MTS raw inputs followed by the use of a CNN.

The best result in the Kaggle competition was 98.01% in the private dataset (i.e., internal dataset used for the final ranking of competitors) and 97.18% in the public test dataset (i.e.,

Study	Method	HAR Dataset Type	ACC (%)
Romera-Paredes et al. [2013]	OVO SVM Ensembling Voting	FE	96.40
Anguita et al. [2013]	OVA SVM	FE	96.37
Kaden et al. [2013]	Kernel variant of LVQ	FE	96.23
Ronao and Cho [2016]	tFFT + CNN	RTD	95.75
Jiang and Yin [2015]	DFT + CNN	RTD	95.18

Table 3.2: Most relevant studies using the UCI HAR dataset with 21-9 working setup.

the one used by competitors for testing and development). However, results from these highly problem-dependent architectures are not comparable with results from studies presented in Table 3.2 since the train and test partition of public and private datasets is not equal to the original 21-9 working setup.

Similar concern is applied to Ignatov [2018] reporting an accuracy of 97.63% but it is not clear the type of partition considered by the author. After running the available code in his GitHub repository, we not only observe that the test dataset contains an excessive number of examples, namely 2993, but also the number of examples per class is different from the canonical UCI partition. Other works with variants of this dataset include Shaohua Wan et al. [2020], Qin et al. [2020], Slim et al. [2019]. As such, based on Table 3.2, we consider the current state-of-the-art accuracy, for studies that maintain the original UCI 21-9 working setup unchanged, stands at 96.40%.

This is a pure MTS classification problem and all the raw data is ready in the right format to feed the models (*Examples* $\times$ *TimeSteps* $\times$ *Variables*). The data is well centered, well distributed, with no significant outliers, normalized with all indicators in the same order of magnitude, and no missing values. As such, no important FE was made.

3.2 Air Pollution - $PM_{2.5}$ Concentration

An air quality dataset retrieved from UCI repository is used for the second case study. With the development of the economy and population all over the world, most metropolitan cities are experiencing elevated concentrations of ground-level air pollutants, especially in fast developing countries like India and China. Exposure to air pollution can affect everyone, but it can be particularly harmful to people with heart diseases or lung conditions, elderly people, and children. Studies show that long-term exposure to fine particulate air pollution or traffic-related air pollution is associated with mortality rates, even at concentration ranges below the standard annual mean limit value [Wang et al., 2018, Cohen et al., 2017]. Therefore, building an early warning system, which provides precise forecast and also health alerts to

local inhabitants, will provide valuable information to protect humans from damage by air pollution. Understanding the behavior of air pollution is needed to predict it and then to guide action to ameliorate it. Currently, three major approaches are used to forecast real-time air quality: simple empirical approaches, advanced physical-based approaches, and ML approaches.

Simple empirical approaches like persistence and climatology methods are based on assumptions or hypothesis; that is, thresholds of forecasted meteorological variables can indicate future pollution level [Dye, 2003]. They are computationally fast but have low accuracy and are primarily used as references by other methods. Advanced physical-based approaches like chemical transport models simulate the formation and accumulation of air pollutants by a solution of the conservation equations and transformation relationships among the mass of various chemical species and physical states. They can provide valuable insights for understanding pollutant diffusion mechanisms. But they are computationally expensive, demanding reliable meteorological predictions, and tuned by a high level physical chemistry experts [Zhang et al., 2012].

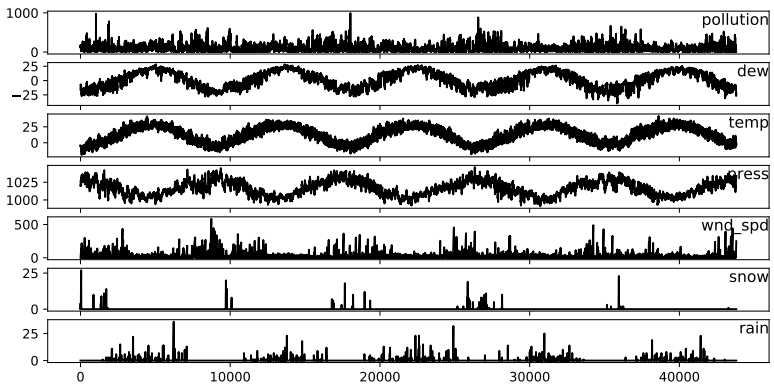


Figure 3.2: 5 years of the MTS used to predict the air pollution - 43800 time steps, one per hour.

Some ML methods have been applied to predict the air quality. Widely used methods include classical ARIMA [Kumar and Jain, 2010], SVM methods [Saxena and Shekhawat, 2017, Vong et al., 2012], simple ANNs methods [Russo et al., 2015, Karatzas et al., 2017], and DL LSTM-based methods [Pardo and Malpica, 2017, Li et al., 2017]. More lately, DL methods based on CNN were also applied to this subject [Sun et al., 2020].

The dataset used in this dissertation reports on the weather and the level of pollution each hour for five years at the US embassy in Beijing, China [Beijing US Embassy, 2014]. Data

include the date-time stamp, the $PM_{2.5}$ concentration pollution indicator, and the weather information including dew point, temperature, pressure, wind direction, wind speed and the cumulative number of hours of snow and rain. The complete feature list in the raw data is as follows:

1. $PM_{2.5}$ concentration;
2. Dew Point;
3. Temperature;
4. Pressure;
5. Combined wind direction;
6. Cumulated wind speed;
7. Cumulated hours of snow; and
8. Cumulated hours of rain.

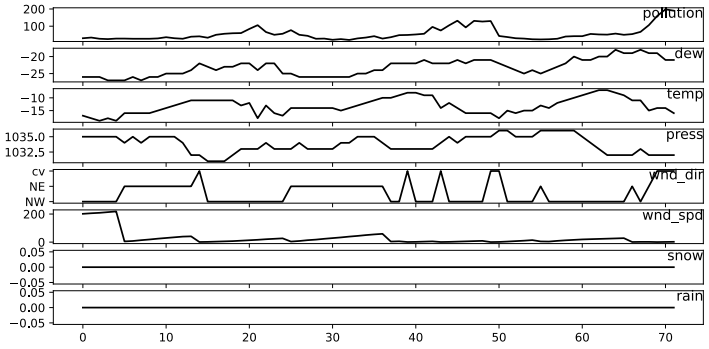


Figure 3.3: Inputs of a training example for pollution prediction. 72 time steps × 8 variables. The target class for this example is level o pollution 1.

The data used in this case study frame a forecasting problem where, given the weather conditions and pollution for prior hours, the level of pollution forecast is made for the next $t + m$ hours.

Feature Engineering

At the input level, minor manual FE is made. It is performed hot encoding of the wind direction into a possible set of four values $\{0.0, 0.33, 0.66, 1.0\}$, normalization of all other variables, and dimensionality transformation for model input compatibility. This dimensionality transformation consists of defining a sliding window with size n over the entire dataset and construct a bi-dimensional feature map of *TimeSteps* × *Variables*. For the model input, we use a sliding window of $n = 72$ time steps (i.e., 72 hours) with the 8 input

variables. Each time a frame advances 12 time steps, a new training sample example is constructed. Figure 3.3 shows one input example for this dataset.

Classification Analysis

At the output level, because we want to work with classification, a frequency analysis of the variable to predict is performed. This is the first indicator itself, pollution level, i.e., $PM_{2.5}$ concentration. The training example output is filled with a value representing the pollution level 12 hours ahead. So the average of 12 time steps ahead is transform into 5 categorical levels of pollution.

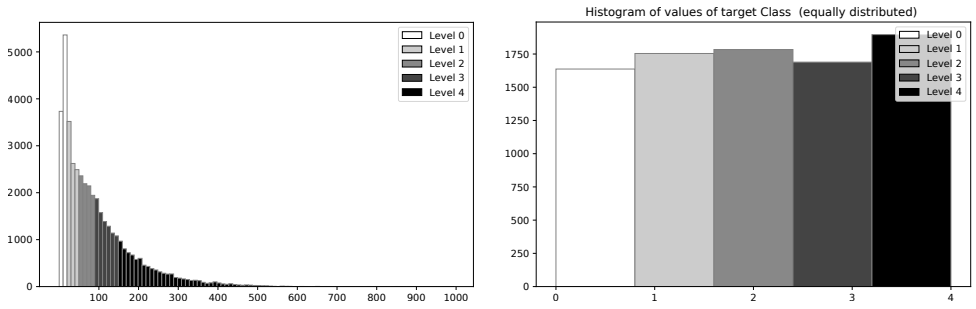


Figure 3.4: Output conversion to obtain an equalitarian number of examples per category.

Figure 3.4 shows how the five classes value intervals are defined to obtain a balanced training dataset by class:

Level 0 $PM_{2.5}$ [0, 18 [
Level 1 $PM_{2.5}$ [18, 48 [
Level 2 $PM_{2.5}$ [48, 89 [
Level 3 $PM_{2.5}$ [89, 153[
Level 4 $PM_{2.5}$ [153,1000]

3.3 Individual Household Electric Power Consumption

The accurate prediction of electric energy consumption in the residential sector is a desirable action to ensure the minimization of potential losses and the maximization of social welfare. A considerable number of studies attempts to extract features from energy consumption data and predict electric energy consumption in the residential sector [Khan et al., 2020, Kim and Cho, 2019, Fumo and Biswas, 2015]. In general, three types of models are used for this

purpose: statistical or econometric [Chujai et al., 2013, Kaur and Ahuja, 2017], classical ML [Amber et al., 2015] and DL methods [Kim and Cho, 2019]). Although statistical approaches are relatively easy to implement and allow to eliminate covariates without explanatory power on the target, they do not avoid the risk of multicollinearity [Kim and Cho, 2019]. Moreover, researchers may have a concern with endogeneity, spurious correlation, omitted variable bias and reverse causation [Wooldridge, 2016]. This endogeneity was confirmed by Fumo and Biswas [2015] that predicts the electric energy consumed in the residential sector through a multiple linear regression models and confirm that the time resolution chosen drastically affects the predictive performance. E.g., if one works in low resolution and tries to predict the nest step, this will be very similar to the previous one. The correct framing of the problem is important to build a model that extracts meaningful information.

Kim and Cho [2019] have put efforts into developing DL models to predict energy consumption using the same datset described in this Section. They propose a DL model that combines CNN and LSTM layers to learn spatial-temporal features of electric energy consumption. A CNN layer extracts relevant features among the set of inputs and respective outputs are used as inputs in the subsequent LSTM layer. The output of the LSTM layer is passed into a dense layer, which generates predicted values of energy consumption (i.e.,in regression mode). The authors also modify the time resolution to understand whether their main result is robust and provide a variable of importance analysis for input attributes. The authors show that their CNN-LSTM model is more capable of capturing irregular trends of power consumption compared to more classical approaches, while simultaneously allowing to identify inputs with a significant effect on the output. Unfortunately, this article does not provide information about the size or part of the data considered for test purposes. Also it is claimed that "*A total of 25,979 missing values were removed for preprocessing*" which perverts the premise of analyzing uniform time intervals of data, crucial for the type of models described. In case of missing values, the best approximation possible should be considered. In this dissertation we filled the missing values with the value at the same time from the previous day.

Retrieved from the UCI ML repository, the present analysis uses the dataset containing information about the electric energy consumption of a home located in France during a period of approximately four years, gathered between December 2006 and November 2010 [Hebrail and Berard, 2012]. It contains measurements of electric power consumption with a one-minute sampling rate. Different electrical quantities and some sub-metering values are available in the dataset. The 7 variables available in the dataset are (as described in the UCI website):

1. Global active power: household global active power (in kilowatt);

2. Global reactive power: household global reactive power (in kilowatt);

3. Voltage: minute-averaged voltage (in volt);

4. Global intensity: household global current intensity (in ampere);

5. Sub metering 1: energy sub-metering No. 1 (in watt-hour of active energy). It corresponds to the kitchen, containing mainly a dishwasher, an oven and a microwave (hot plates are not electric but gas powered);

6. Sub metering 2: energy sub-metering No. 2 (in watt-hour of active energy). It corresponds to the laundry room, containing a washing-machine, a tumble-drier, a refrigerator and a light; and

7. Sub metering 3: energy sub-metering No. 3 (in watt-hour of active energy). It corresponds to an electric water-heater and an air-conditioner.

Feature Engineering

DL NN models employed in this study are trained and tested considering a 80%-20% partition rule. As required by MTS problems, shuffling of samples is not used. Figure 3.5 clarifies that the time resolution of inputs (outputs) is defined in hours (days), respectively. In particular, 168 hours time steps are used as inputs, while the output corresponds to the average of the next 24 hours time steps. As such, when configured for regression, DL NN developed models could be comparable with alternative options that consider a measurement of the output in days.

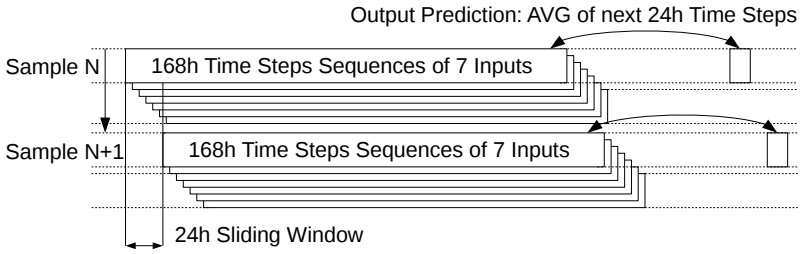


Figure 3.5: Sample format used for the developed models.

Hancock et al. [1988] and Hamilton et al. [2007] highlight the need to normalize data in cases where inputs have different units of measure. In addition, outliers can substantially affect the learning ability of NN models. A relevant characteristic of learning algorithms is that they tend to smooth out noise, which allows to effectively model noisy systems. However, if data are concentrated in a very small portion of the input range, then explanatory variables subject to this type of bias may not have a significant impact on learning and final predictions

of the model. Deboeck [1994] proposes a solution based on FE to mitigate this concern, which consists of applying a bilateral truncation to the MTS values based on frequency analysis and histogram re-scaling.

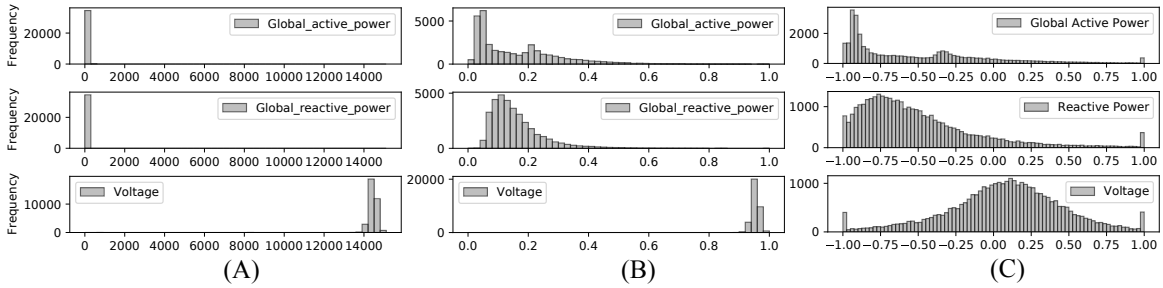


Figure 3.6: Data treatment possibilities: (A) raw data, (B) standardization, (C) histogram re-scaling.

Figure 3.6 presents three data treatment possibilities. Panel (A) shows the distribution of output and input variables without normalization, panel (B) presents the distribution of output and input variables with standardization and panel (C) clarifies the distribution of output and input variables with normalization based on frequency analysis and histogram re-scaling. Data presented in panel (C) concentrate values between -1 and 1 that are used in the subsequent analysis.

Classification Analysis

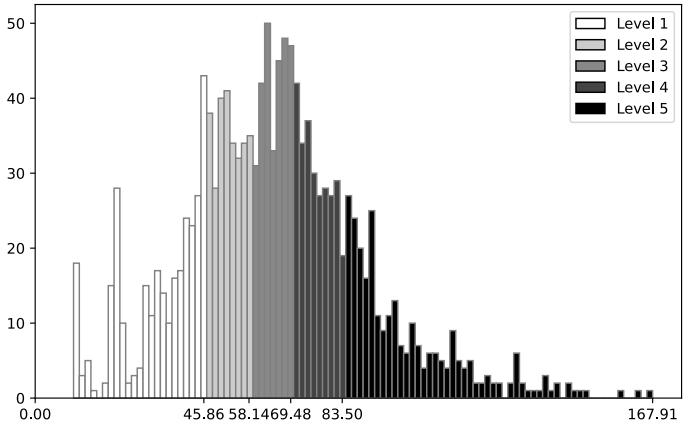


Figure 3.7: Definition of output classes. Levels of energy consumption.

At the output level it is shown a transformation of this problem into a classification problem, according to electric energy consumption levels, to maintain compatibly with the global framework of this dissertation. However, for this case study in particular it is also provided more detailed information, in Chapter 4, about the robustness of the best model configured

for regression. Knowing that a frequency analysis is used at the output level to impose a classification problem, Figure 3.7 clarifies that an egalitarian distribution in the number of examples is considered for each qualitative class. Five output categories are defined, which correspond to different consumption levels of electric energy.

3.4 Betting Exchange Markets

Modeling exchange markets requires to consider a market as a platform in which people and entities can trade fungible items of value, with low transaction costs, at prices that reflect supply and demand. The extracted models consists of a representation of the multiple interactions among participants. As in most modeling problems, unexpected external events can override the assumptions on which the system's predictive behavior relies. The described framework concerns with purely speculative markets, thus, the system considered here is based on a closed-loop interaction, i.e., forecasting is exclusively dependent on market data itself.

One of the best exchange markets compliant with the mentioned closed-loop interaction property found in the real world are markets present in betting exchanges. A betting exchange is an entity that offers trading services to buy and sell bookmaking contracts. These contracts are structured as binary options (i.e., win or lose) where the payoff is either some fixed amount of money or nothing at all, depending on the outcome of a future event [Chen et al., 2008, Schumaker et al., 2010]. Betting exchanges trade heavily on sports events but also offer markets on elections and other types of events. In analogy to the financial markets, the buy and sell operations are replaced by betting for and against (i.e., Back and Lay). The presented methodology can be applied to exchange markets that provide market depth access, the so-called level 2 market data. Examples include futures (e.g., Dorman Trading, Phillip Capital), forex (e.g., FXCM), securities (e.g., Euronext Bonds), betting exchanges (e.g., Betfair, Betdaq, Matchbook), and cryptocurrency (e.g., Coinbase, Bitmex). All of them share the same functional basis; therefore, they can be adapted into our framework. Some other types of exchanges do not provide market depth data (e.g., CFDs - exchange virtualization) and can not be considered.

Table 3.3 shows a snapshot example of a market depth view. This information is referred throughout the manuscript as a Raw Data Frame (RDF). The "Price" column describes the ladder of possible transaction prices. The market buys and sell amounts are listed in the "Bid" and "Ask" columns. The "Buy" and "Sell" columns represent the agent's own orders waiting to be matched. When the buy and sell orders reach the same price, there is a matched amount transaction. The amount transacted at each price is listed in the "Volume" column. The yellow cell shows the last matched price.

Buy	Bid	Price	Ask	Sell	Volume
		⋮			
		5,1			20
		5,0	250		93
		4,9			68
		4,8	263		24
		4,7	148	10,00	70
		4,6	349	5,00	76
	8	4,5			217
	2	4,4			23
10,00	10	4,3			4
	448	4,2			
	398	4,1			
	335	4,0			
		⋮			

Table 3.3: Snapshot of market depth RDT information.

Betfair exchange is considered as case study since it provides an easy and free Application Programming Interface (API) access to obtain markets raw data. Bets are sold and bought at different prices, which can also be referred to as odds. The price dynamics allows to achieve Profit & Loss (PL) before knowing the event's final outcome. Depending on the investor's beliefs, a price may move just a few or many ticks [1]. Thus, price volatility is shared knowledge. The UK To-Win horse racing market is characterized by high liquidity and volatility levels, constituting a decisive factor for its selection to meet our research scope.

The chosen moment to act in the market is 10 minutes before starting the race, i.e., pre-live. The reason for this choice is straightforward: before the beginning of the horse race, the price of each runner is highly subject to speculation. This moment of action does not depend on external factors but rather on the information available in the market itself, thus constituting a strongly closed-loop system. As such, given this atomistic property, this study is exclusively concerned with purely speculative markets.

10 minutes before the race start is when intense activity on the market starts to happen. Fig. 3.8 exposes the average value of trading volume, liquidity (i.e., sum of all runners amounts waiting to be matched - at the bid and ask price only), and volatility (i.e., number of ticks variation in absolute value per minute) for the complete sample of observed races, considering all runners involved in a given race. From the 10th minute before each race starts until the effective start of each race, the average trading volume increases about 4.2 times, the average liquidity increases approximately 3.4 times, and the average volatility increases

[1] The unit of measure of a price change is designated by tick.

about 1.6 times. Hence, from a dynamic point of view, we observe that all variables increase as we approach the beginning of a race.

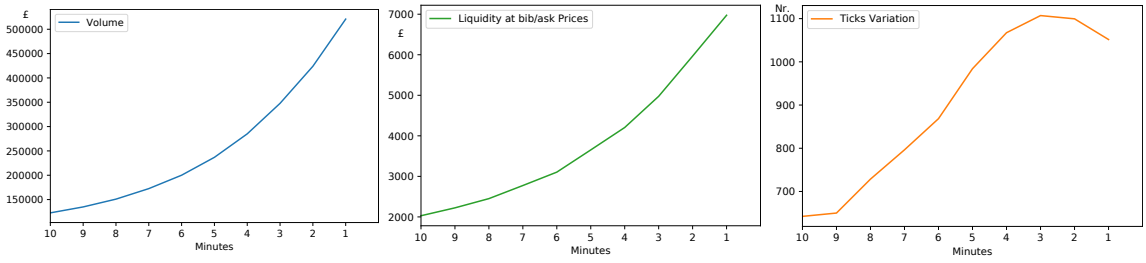


Figure 3.8: Average trading volume, average liquidity at the bid and ask price, and average number of ticks variation in absolute value per minute.

Raw data were collected directly from Betfair servers in real-time at a 2 frames/samples per second rate from the 1st September 2014 to the 29th August 2016. For the sake of brevity, summary statistics with respect to volume, liquidity, and volatility at the race level on a per-minute basis during the 10 minutes time window before the start of each race are provided in Table A1 of the Appendix. Globally, we recorded a personalized database related to the UK To-Win horse racing markets characterized by 15 to 30 daily races with 3 to 25 competing horses per race.

We use 8 minutes of information to predict the last 2 minutes price variation of runners before the race starts. This type of trading, with a short time of exposure that gives primordial importance to the market data itself, is normally performed by day traders and called "*day trading*". Also, when performed automatically, the execution of very sort-term trades can be called high-frequency trade. The methodologies implemented in this thesis can be framed in these groups. Access to the market depth is fundamental in high-frequency trading or *day trading*. In contrast with long-term trading, the information about what happens in detail at different prices, transacted one and around, in every second, is very important. To access this type of information, the trader (or agent) must have a "*ladder view*" of the market (see Table 3.3). All the data used in this case study for price forecasting is present on the evolving price ladder.

In order to archive some PL prices need to move up or down. Prices fluctuate due to the supply and demand law. If demand for a promising company stock is high, the shares price will rise. If there are a lot of sellers, prices will drop. The same applies to betting exchange markets. If there is most of the participants Lay on a runner (e.g., horse in a race or team in a football match), the fixed odds for winning the event will drift up. If bettors stats to invest in favor of a runner to win, it will cause the odds to decrease. Depending on the force

and investor's believes, a price or odd may move just a few ticks or a lot more.

3.4.1 Trading Framework

As betting markets are short-lived, yield easily quantifiable final payoffs for the assets traded, and have a degree of repetition, they provide clean tests of efficiency. In betting exchanges, there are events, e.g., tennis match or horse race. On each event there are runners , e.g., horses in a horse race. On runners, Back and Lay bets are placed. A Back option is a bet on the runner to win, while Lay is a bet on the runner to lose. Bets are placed at a given price. For instance, the price 2.0 is a 50% of chances ($1/2 = 0.5$), price 1.01 is a 99% of chances ($1/1.01 = 0.99$), 100 is a 1% of chances ($1/100 = 0.01$). In Table 3.3 we have the following bets placed (but not matched yet):

- Lay of £10.00 at 4.30 (Lay 10@4.3);
- Back of £10.00 at 4.70 (Back 10@4.7); and
- Back of £5.00 at 4.60 (Back 5@4.6).

If a bet is placed at one price that "the market" is willing to buy, the bet will be matched at the best price offered. For example, in the market state of Table 3.3 if one Back bet 15@4.4 is placed (on the blue side) it will match £8.00 at 4.5, £2.00 at 4.4, and will leave the remaining £5.00 at 4.4 unmatched on the ask side waiting for someone to buy with a Lay bet. The traded volume information will have its update. This is how the prices move in the market. Since this bet was matched in two ($N = 2$) different prices, the global matched price of this bet can be calculated using equation 3.1.

$$Price\ Average = \frac{\sum_{n=1}^{N}(Price_n \times Amount_n)}{\sum_{n=1}^{N}(Amount_n)} \tag{3.1}$$

If a Back is placed above the best offer in the market (4.5 in Table 3.3), for example, 15@4.9, it will stay in the market unmatched and, so for, waiting to be matched in a First In First Out (FIFO) queue of all back orders on that price. The same happens to a Lay bet if it is placed at a lower price than the best offer (i.e., counter bet waiting to be matched). Only unmatched or partial unmatched amount of bets can be canceled.

The profit of a Back bet is calculated using equation 3.2 and the liability (i.e.,in case of loss) of a Back bet is the amount of the bet itself.

$$Profit\ Back\ Bet = Amount\ Back \times (Price\ Back - 1) \tag{3.2}$$

The liability or amount in case of loss of a Lay bet is given by equation 3.3 and the profit is the amount of the bet itself. In resume, Lay is the mirror of Back.

$$Liability\ Lay\ Bet = Amount\ Lay \times (Price\ Lay - 1) \tag{3.3}$$

Using Back and Lay combinations, it is possible to assure a fixed PL before the end result of an event. Example of a trade where it does not need to know the result of an event to have a fixed PL:

- Back of £2.00 at 2.12 (Back 2@2.12) Matched; and
- Lay of £2.00 at 2.10 (Lay 2@2.1) Matched;

For a bet to be matched, it must become the best offer in the market and it has to be purchased with a counter bet.

When the runner is a winner, then the profit (Back) - loss (Lay) is: $2 \times (2.12 - 1) - 2 \times (2.10 - 1) = 2.24 - 2.20 = 0.04$

When the runner is a looser, then the profit (Lay) - loss (Back) is: $2 - 2 = 0$

Notice that if we have this kind of Back and Lay bet combination with the same amount at different prices, there will be profit (if the Back price is higher than the Lay price) or loss (if the Back price is lower than the Lay price) only if the runner in question wins the event. If any other runner wins the event and the combination of Back/Lay bets have the same amount, the PL will be 0. The agent can distribute the guaranteed PL in one runner across all other runners. To distribute the PL equal for all outcomes the amount to close the trade bet must be recalculated. This process is called "*do the greening*" or "*hedging*". If a Back position is open on the market, the amount to close the position with the corresponded Lay bet is calculated using equation 3.4.

$$Close\ Amount\ Lay = \frac{Price\ Open\ in\ Back}{Price\ Lay\ to\ Close} \times Amount\ Open\ in\ Back \tag{3.4}$$

If a Lay position is open on the market, the amount to close the position with the corresponded Back is calculated using equation 3.5.

$$Close\ Amount\ Back = \frac{Price\ Open\ in\ Lay}{Price\ Back\ to\ Close} \times Amount\ Open\ in\ Lay \tag{3.5}$$

Software

The developed software framework is described as follows. It is an event-based architecture [Zweigle et al., 2010, Kefalas et al., 2009, Deugo et al., 2001]. The connections between modules are made through interfaces based on architecture patterns promoting the production,

detection, consumption of, and reaction to events. Figure 3.9 illustrates the main modules and their connections. Parallel-processing approaches [Magee et al., 1994, Yau et al., 1995] were also applied, in the sense that it is possible to instantiate several *Trading Agents*, with different policies, running in parallel, managing several trades simultaneously. Next, it will

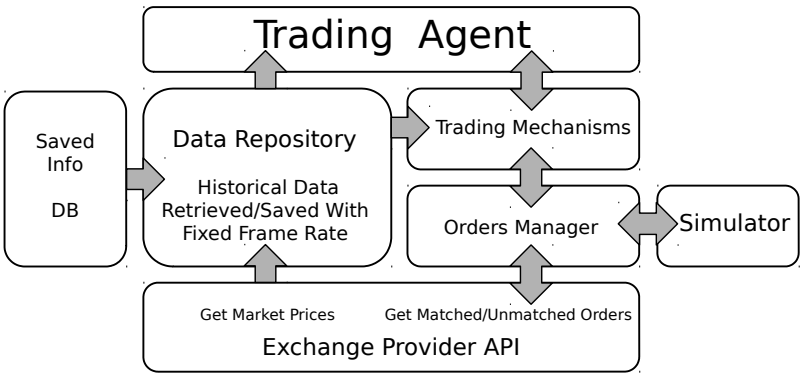


Figure 3.9: High level architecture for automated betting exchange.

be provided a brief description of each module.

- **Exchange API**

We use Betfair [Betfair, 2012, Pitt et al., 2005] exchange API for the work presented in this dissertation. Betfair API allows software developing companies to access Betfair data. In this case Betfair is a service provider and other companies are clients to its services. Usually, it is used to develop trading software or software for tipsters. Its goals are speed and manageability. Betfair API can also be used to develop autonomous agents. Betfair API is used as low layer communication between our framework and the exchange server. For the software framework described here, this low level layer interacts with the *Data Repository* module, providing data about the prices and volumes of each runner, and the *Orders Manager* module, providing data about the states of the bets, and also for placing and canceling bets interactions. The main Betfair API services [Betfair, 2012] used for these actions are:

- *Data Repository*
  - Get Complete Market Prices; and
  - Get Market Traded Volume.
- *Orders Manager*
  - Get Matched and Unmatched Bets;
  - Place Bets;

  - Update Bets; and
  - Cancel Bets.

Depending on the type of licensing, there are different number of calls permitted per minute for each service.

- **Data Repository**

The *Data Repository* module is responsible for data gathering, inform listeners about new data or new states of the market, save the data, and replay saved data.

The main interface of this module is shown on Listing 3.1. The *Market Update* event type simply informs listeners that new data about the runners prices and volume has arrived. It is possible to trade either before or after one race starts. Thus, we have the event type *Market live* that is activated when the market becomes in-play. The *Market Suspended* event type is activated when the market is suspended. The market can be suspended for different reasons depending on the type of market, e.g., in a soccer match the market is suspended after a goal until the game restarts. Also, right after the markets become in-play, they are suspended for a short time. The *Data Repository* module can be instantiated with a new market by an external object. When this happens, the *Market New* event is delivered.

Listing 3.1: Market Change Listener Interface

```
1 public interface MarketChangeListener {
2
3       public enum EventType {MarketUpdate, MarketLive, MarketClose, MarketSuspended,
4                                           MarketNew }
5
6       public void MarketChange(MarketData md,
7                                   MarketChangeListener.EventType marketEventType);
8 }
```

This module can be connected to the Betfair server through the Betfair API or saved files to be used for data replay and simulation, as explained later in this Section.

- **Orders Manager**

The *Orders Manager* module assures the correct treatment of bets information and manages all objects with a *BetListener* interface (i.e., *Trading Mechanisms*) informing them about the state of their bets. It is important to centralize all the bets processing to optimize the number of calls to the *Get Matched and Unmatched Bets* service, which is limited. Also, sometimes the Betfair API does not return the ID of a placed bet, leaving the program unclear about

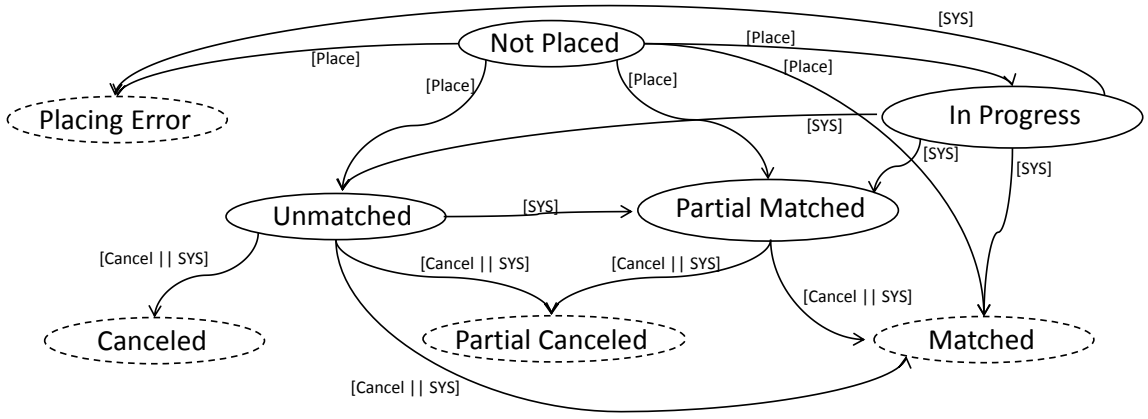


Figure 3.10: State machine for an order.

the placement. In this cases, the *Bets Manager* module tracks the bets without "owner" and re-assigns them correctly (see Figure 3.10 state *In Progress*). Figure 3.10 represents the possible states and transitions of a bet in the framework. $[Place]$ and $[Cancel]$ transitions are made by the agent or *Trading Mechanism* modules (i.e., client side active action). The $[SYS]$ is made by the system, e.g., the system automatically changes the state from *unmatched* to *matched* when the order is filled. This state machine is the base and heavily used by the *Trading Mechanisms* module to control the market's agent position.

- **Trading Agents**

To instantiate a *Trading Agent* object, it has to be extended to the abstract superclass "*Bot*". This class implements all the interfaces and virtual methods to interact with the *Data Repository* and *Trading Mechanisms* modules. The *Trading Agents* objects are normally attached to one market observing one runner but is possible (and useful, e.g., for *dutching* and *bookmaking* techniques) to make these objects observe several markets and runners simultaneously. The *Trading Agent* object can also initialize *Trading Mechanisms* objects, i.e., trading processes, whenever it takes some conclusion about a runner forecasting. When a *Trading Mechanism* starts, running in parallel, the *Trading Agent* is informed about the state of the trade along the execution. On top of this high-level object, it becomes easy to implement decision policies interacting with the markets, since simple rule-based decisions policies to more complex methodologies, e.g., time series predictions.

- **Trading Mechanisms**

The *Trading Mechanisms* are used in some way to discipline the trader attitude towards the market. In other words, these methods are likely to be implemented on a computer. They

are executed after a decision on the market forecasting. Once the decision for depreciation or appreciation of a runner is taken, a sequence of steps is started in order to maximize profit. In this dissertation, we will focus on three of these trading methodologies:

- Scalping;
- Swing; and
- Trailing-Stop.

In the framework, these methods are implemented on the *Trading Mechanism* module (see Figure 3.9). After one *Trading Agent* parametrizes and instantiates one *Trade Mechanism*, it will run in parallel and will inform the owner agent along the way about the state of the trade. Ultimately it will inform the agent that trade is over and the PL of the operation.

- Scalping

Scalping is used for very short-term trading. A scalping trader looks to make lots of small profits, which in time add up. Scalping relies on lots of active participants in the market. Scalping works better in markets with lots of liquidity. The concept is simple: if a Back bet is placed at a certain price, a Lay bet must be placed right in the next lower price, or the other way around for the opposite direction to make profit. The PL is equal to the difference, or spread, between the Back and Lay price as explained in Section 3.4.1. The Betfair betting exchange is an ideal place to trade in this way. Mainly in horse racing because there is lots of liquidity in these markets, in particular just before the start of the race. Scalping the market means trading in the market tick by tick. One tick is one step in the prices scale of the ladder. For example, if a Back at 2.12 is placed, one successful scalp will close the position with Lay at 2.10 (i.e., one tick down). If a trade starts with a back, it means that the price was predicted to go down. If it is predicted to go up, the scalp starts with a Lay bet.

Figure 3.11 represents the state machine used to process one Back⇒Lay scalping (i.e., prediction for the price to go down). One Lay⇒Back scalp will be a "mirror" of this state machine. The Price Back Request (PBR) is the price where the agent enters the market, while the Price Back Now (PBN) is the price at the current moment. If the price has already moved (i.e., $[PBR \neq PBN]$) when the order reach the scalp module (i.e., the start state), it will assume the opportunity was lost (i.e., the prices already went down) or the prices went in the wrong predicted direction, so it ends the process without doing nothing. Otherwise, $[PBR == PBN]$ opens position on the market with a Back bet. After the bet is ordered to be placed, if the bet was not matched after some time, it will end the trade (canceling

the bet) because it will assume the prices already move down, and the opportunity was lost. Otherwise, it will try to close the position placing a Lay bet. If the price goes one tick down, it will close the trade with profit. If the price does not move, it will wait some time. After that time is over and the close bet was not been matched, it will try to close at same price with null PL. If the price goes up, it will close in "emergency" with loss.

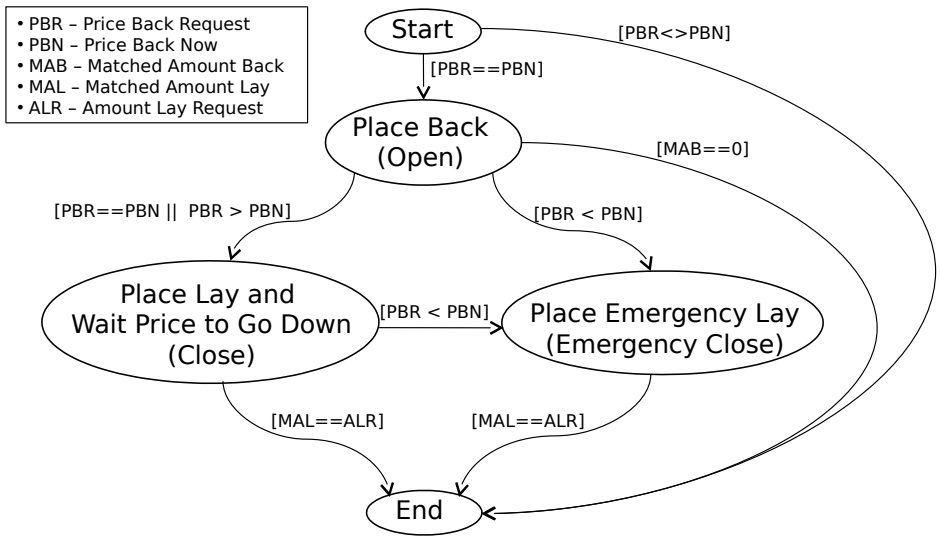


Figure 3.11: Simplified graph scheme for a Back-¿Lay scalp implementation.

Listing 3.2 presents the declaration of the constructor method of the object that runs the scalping process in parallel. The *MarketData md* argument identifies the event (e.g., horse race, soccer game, tennis), the *RunnersData rd* identifies the runner to be traded. The *double entryAmount* initial stake of the trade and *double entryPrice* is the entry price to open position on the market. The *int waitFramesNormal* is the number of actualizations received from the *Data Repository* before it tries to close at the same price he entered (i.e., with null PL). After the *int waitFramesNormal* expire, the *int waitFramesEmergency* also starts a count down and, when expiring, it will close at the best offer available to place the counter bet (i.e., *emergency* close with loss). The *Bot botOwner* is the owner agent of the trade, used to be informed about the state of the scalp. Finally, the *int direction* argument indicates the predicted direction of the price movement.

Listing 3.2: Main parameters for Scalping mechanism

```
1 public Scalping(MarketData md,
2         RunnersData rd,
3         double entryAmount,
4         double entryPrice,
5         int waitFramesNormal,
6         int waitFramesEmergency,
7         Bot botOwner,
8         int direction, ...);
```

- Swing

The swing methodology is very similar to the scalping. The main difference is the number of ticks the price has to move in order to enter the close state [Carter, 2007]. On the swing methodology it is possible to define the offset number of ticks to close in profit and the offset number of ticks to close in loss. If the price stays inside this interval offset, it does not do nothing. Swing with offset of 1 tick for profit and offset of 1 tick for loss is the same as scalping.

Listing 3.3 describes the constructor for the swing process initialization. Besides the same parameters present on the scalping constructor, there are the *int ticksUp* and *int ticksDown* representing the offset number of ticks to close in profit and loss, which depend on the *direction* parameter. There is also the *boolean* $frontLine$ and *int* $waitFramesOpen$ new arguments. These are used when the agent does not want to enter the market where offer is available, but wants to wait until the market reaches the price given in the *entryPrice* argument. If $waitFramesOpen$ expires and the market does not match the entry bet, it will cancel the trade process. If $frontLine = true$ it will ignore this time (i.e., $waitFramesOpen$) and assumes the agent wants to enter the market at the *entryPrice* argument where the counter offer is available.

Listing 3.3: Swing constructor example

```
1  public Swing(MarketData Market,
2          RunnersData rd,
3          double entryAmount,
4          double entryPrice,
5          boolean frontLine,
6          int waitFramesOpen,
7          int waitFramesNormal,
8          int waitFramesEmergency,
9          Bot botOwner,
10         int direction,
11         int ticksUp,
12         int ticksDown);
```

- Trailing-Stop

The trailing-stop methodology is used when the agent is looking to catch a broader trend in a market but wants to retain a stop loss condition if the trend starts to turn. The concept is simple, after a position is open in the market the close bet is set to close with a tick offset behind, and moves only when the price moves in the predicted direction. Eventually, the price will move in the reverse direction reaching the close price and the order is placed to close the trade.

Figure 3.12 represents the state machine used to process this methodology for the price prediction to go down (i.e., Back⇒Lay). The Price Lay to Close - PLC represents the dynamic changing price N ticks above the PBN. The state "*Update PLC N Ticks Above PBN*" performs the price update repeatedly when $[PBP > PBN]$, i.e., the runner price moves in the predicted direction, for this case, down. The PLC is updated only when PBN goes in the predicted direction. When the price turns direction, eventually it reaches the PLC. When it reaches, the close order (i.e., Lay bet) is placed. Then, when $[MAL = CAL]$, it means the close bet is completely matched. The price went in the reverse direction (i.e., up) filling completely the close bet and closing the trade.

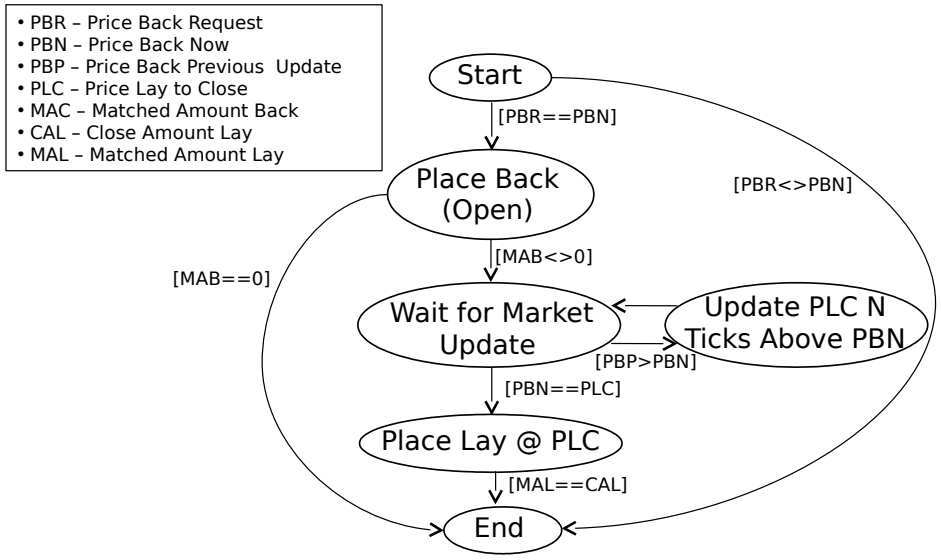


Figure 3.12: Simplified graph scheme for a Back⇒Lay Trailing-Stop implementation.

Listing 3.4 is the the constructor method of the object that runs the trailing-stop process in parallel. The $offset$ is the number of ticks offset to follow the runner price. Also, we have included a number of frames update *waitFramesNormal* to close after a given time to control the duration of the trade.

Listing 3.4: Trailing-Stop constructor example

```
1  public TrainlingStop(MarketData Market,
2          RunnersData rd,
3          double stakeSize,
4          double entryPrice,
5          boolean frontLine,
6          int waitFramesOpen,
7          int waitFramesNormal,
8          int waitFramesEmergency,
9          Bot botOwner,
10         int direction,
11         int offset);
```

- **Simulation**

This Section addresses issues and limitations to achieve simulation in this kind of markets. This process implies the simulation of the bet placement. There are two main problems to simulate a bet placement on the market:

(I) The first main problem is the bet amount influence on the market. The unmatched bets will not appear in the real market and matched bets will not consume or alter the available amounts in the real market. This issue is impossible to bypass since, in simulation, the actual amount of the bets is not placed. For example, this limitation makes it impossible to simulate and test *Trading Agents* relying on *spoofing* methodologies.

(II) The second problem is the simulation of the matching process. The bets of all traders on the market are placed in a *FIFO* queue of bets for each price on the runner. It is impossible (there is no data provided by Betfair API about it) to know in which position in the queue is our bet. It is possible to have an approximate idea by looking to the volume matched in the placement price and monitoring this volume evolution. But since these markets are of very high frequency trade, it is impossible to know the exact volume on the price when the placement order reaches the Betfair server. Also, it is impossible to know if canceled bets were ahead or behind our bet, damaging the process of volume monitoring to try to solve this issue. For the described framework, we assume the worst case possible: bets are considered to be in the front of the queue when the amount of volume transacted is equal to the amount that was in front of the order when it was placed. Also, after that, we control the amount matched of the order with the volume variation. An order is fully matched only when the volume variation reaches the order amount.

3.4.2 Data Collection and Feature Engineering

Raw data was collected on a twice per second basis. There are between 15 to 25 races per day and data is only collected during 10 minutes before a race starts. The goal is to treat the collected data and update the models every month, as shown in Figure 3.13.

The raw data corresponds to the data frames listed in Table 3.3. Then, we transform them into examples used to fit the models. The examples are organized in training and test datasets. The set of examples is constructed from the present to the past until the maximum number of examples defined for the training purpose is reached.

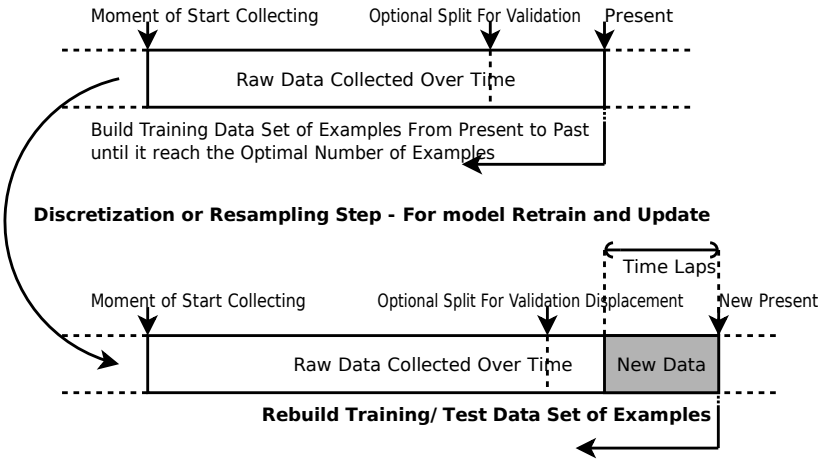


Figure 3.13: Add-in information to the raw dataset for global re-training of the models.

Rule-Base Filtering

This step has required expert opinions on this type of market. Table 3.4 systematizes the decision tree that underlies the specificities of each market dynamics. The combination of the properties listed in Table 3.4 generates 54 different categories (i.e., tree leaves). These are indexed to simplify data treatment. For instance, category 41 corresponds to a market dynamics characterized by the following properties:

$$root/nofavorite/mediumRunners/midleOdd/highLiquidity/ \Rightarrow Model(41)$$

From the 54 categories only 9 satisfy the minimum amount of data required to train the models because only these correspond to the more likely market states. To train a model from scratch, we define the minimum number of examples to be 1200. For the other categories, further studies must be taken concerning the use of transfer learning. The transfer must be done sequentially, from category to category, by similarity.

Inputs and Outputs

For the input, 514 RDFs are used which corresponds to about 8 minutes of data to predict the last 2 minutes price movement before the start of the race. Segments of 4 RDF are compressed into a single value (see Figure 3.15), which leads to the consideration of 128 time steps. This data compression facilitates the computation and attenuates the risk of overfitting. Each race constitute one example for the training. Nine indicators correspond to the model inputs, this leads us to the consideration of the input format : *128 TimeSteps* $\times$ *9 Variables*. Figure 3.14 illustrates 4 examples, i.e., 4 races with the 9 indicators evolution.

Favorite (1)	**Runners** (2)	**Price** (3)	**Liquidity** (4)
Yes	Few	High	High
No	Medium	Medium	Medium
	Many	Low	Low

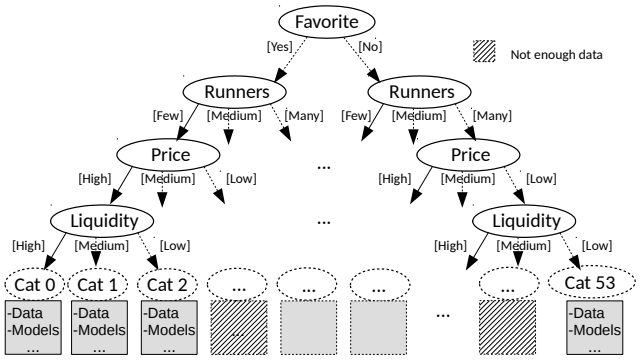


Table 3.4: Rule-based decision tree.

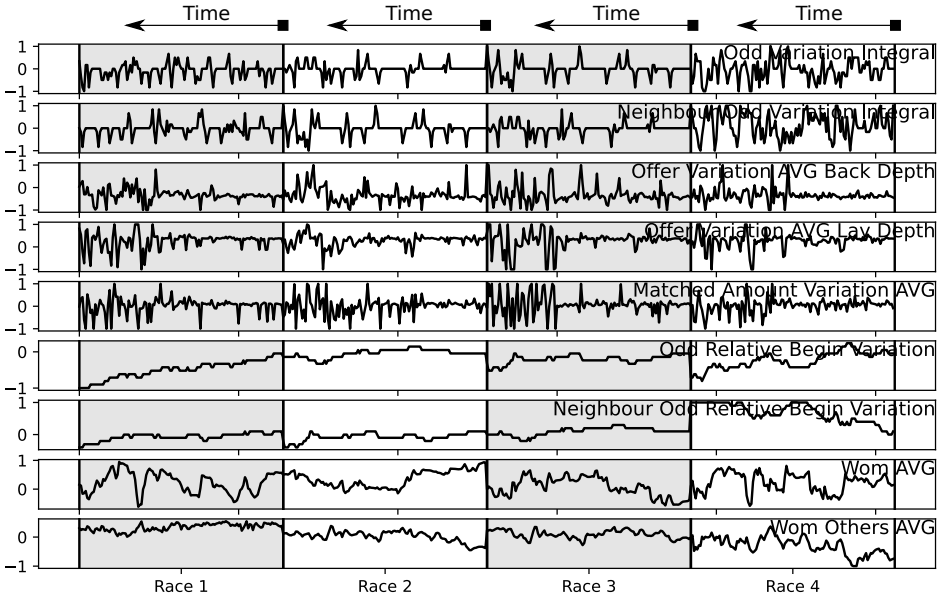


Figure 3.14: 4 races input examples, indicators evolution. Each input sample has 8 minutes of data.

The 9 indicators selected as inputs of the DL NN models are:

1. Integral of the price change of the runner in trade;
2. Integral of the price change of the competitor runner;

3. Liquidity variation in the ask side;
4. Liquidity variation in the bid side;
5. Volume variation and direction;
6. Price variation relative to the beginning of the sequence of the runner in trade;
7. Price variation relative to the beginning of the sequence of the competitor runner;
8. Weight Of Money (WoM) of the runner in trade; and
9. WoM of all the others runners combined;

Figure 3.15 shows an example of the market state evolution corresponding to a 3 RDFs, in discrete time period, compressed into one time segment value. It serves as a show case for the construction of the indicators considered in our case study. The selected indicators are also exposed in Figure 3.15. The first indicator corresponds to the integral of the price change of the runner subject to modeling, being given by the ticks' integration during the time segment. The second indicator corresponds to the integral of the price change of the competitor runner. In our case, the competitor runner is the one holding the closest price. In financial markets, this choice may be given by an expert's opinion, i.e, another market with strong positive or negative correlation. Since this is similar to the previous indicator, its graphical representation is omitted. The third and fourth one corresponds to the amount variation on the Ask and Bid side respectively. Note that, for this example, it is assumed that the fourth past frame is equal to the third past frame. The fifth indicator highlights the *market strength*. It is given by the variation of the matched amounts which provides information about the volume direction and strength. The sixth indicator corresponds to the price variation between the beginning of the entire sequence t_0 and the segment in processing t_i; for this example, in Figure 3.15, we assume this 3^th RDF segment is the fist of the 8 minutes. The seventh indicator is the same but applied to the competitor runner. The eighth indicator is the average WoM of the RDFs in the segment. The WoM is represented by a percentage value and shows when the market is balanced or unbalanced. The market is said to be balanced when the amount of money unmatched on each side of a selection is the same. This means the amount placed on ask side must be approximately equal to that placed on bid side. The underly logic is: when there is more unmatched money on ask side than the bid side the price decrease. The WoM pushes the price down. The same applies the other way around. WoM indicator is given by :

$$WoM = \frac{Amounts\ Bid}{Amounts\ Bid - Amounts\ Ask} \tag{3.6}$$

For this example, in Figure 3.15, we consider only the depth of the best 3 prices around the transacted price. Depending on if the price is high, medium, or low (see Table 3.4), the depth

used is 2, 3, and 4. For the ninth indicator, WoM is also applied but to all others runners. The logic is that if the ladder of the unmatched amounts in all other runner is pressing the price in one direction the runner in trade will be pressed in the opposite direction.

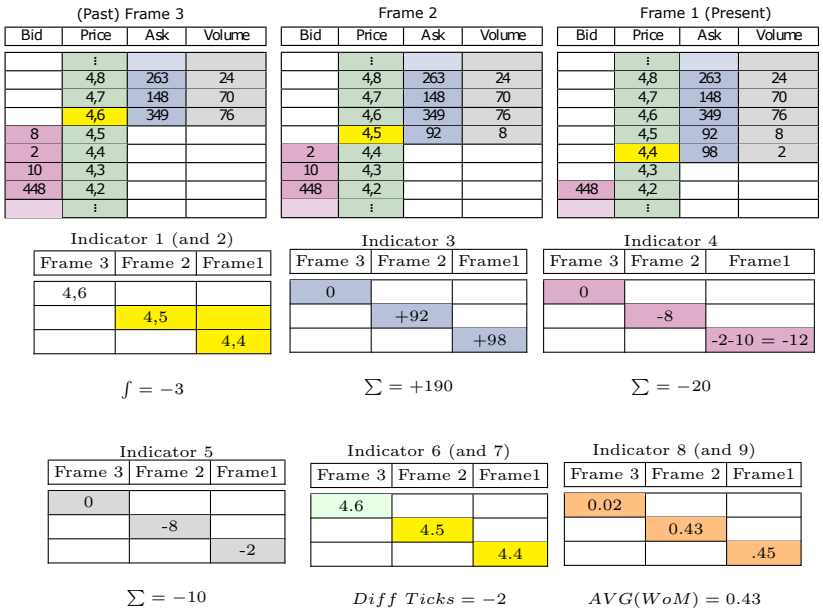


Figure 3.15: Example of processing the 9 indicators given a segment of 3 RDF[2].

Finally, the model output or *target* corresponds to the integral of the price variation, measured in ticks, for the last two minutes before the race starts. By compressing the data from various segments in this way, we define a multivariate time series prediction problem with 128 time steps.

Frequency Distribution Histograms

The same technique, to address outliers, applied for the case study described in Section 3.3 was applied here [Deboeck, 1994]. Figure 3.16 clarifies the automatic process of outliers truncation. Data is normalized into the interval $[-1, 1]$ after a frequency analysis and histogram re-scaling. The rescale of maximum and minimum raw values consists of truncating 10% of the histogram tails, the original examples are altered but not removed. Only then data is fed as input for the models training. This operation is systematically applied to all inputs for each category. Figure A1 in Appendix illustrates this operation result for all indicators used in this case study.

[2]On frame 1 of Figure 3.15 the Lay amount of 10 at 4.3 disappears not due to the matching process but to exemplify a cancellation amount.

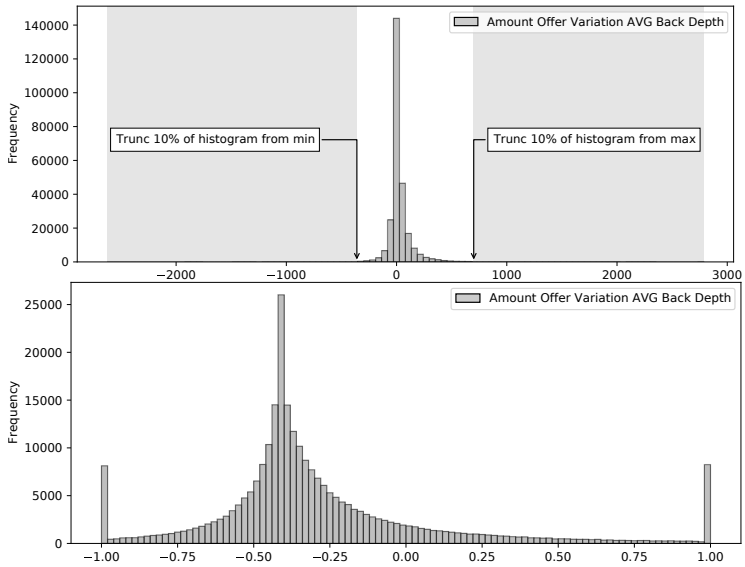


Figure 3.16: Example of histogram re-scaling with truncated tails at 10% level to find min-max values for input normalization. The illustration represents input #3 (Liquidity variation on the ask side) on category #41 of the rule-based system index.

Thenceforth, similar technique of using the histogram for data partition is applied at the output level to transform the regression problem into a classification problem. An egalitarian distribution in the number of examples per qualitative class is obtained in order to avoid overfitting and/or biased models. The output for this case study is the integral , i.e., area, of the tick variation of the runner price, relative to the last 2 minutes before the start. When the integral is hugely negative, a “strong down” price change is considered, and the first qualitative class is established. When the numerical solutions falls into the second qualitative class, a “weak down” price change is considered. When the numerical solution falls into the third qualitative class, there is a “neutral” price change. When the numerical solution falls into the fourth qualitative class, we have a “weak up” price change. Finally, a “strong up” price change occurs when the numerical solution falls into the fifth qualitative class. The classification procedure is clarified in Figure 3.17. This way each class has approximately 20% of examples in the training dataset. A ”strong” movement prediction class suggest an activation of a trailing-stop trading mechanism while a ”weak” movement prediction suggest a small swing activation. The choice of target and stop-loss prices in each category depends on the de-normalization of the output based on the histogram presented in Fig. 3.17. The main idea is that this process allows to adjust the parametrization of the trading mechanisms (target and stop-loss) according to each category. The target price corresponds to the average of the maximum tick variation for all examples of the collected data falling at a given class during the predicting time. The stop-loss is defined as 80% of the target price for swings and 60% of the target price for trailing stop. Table 3.5 presents

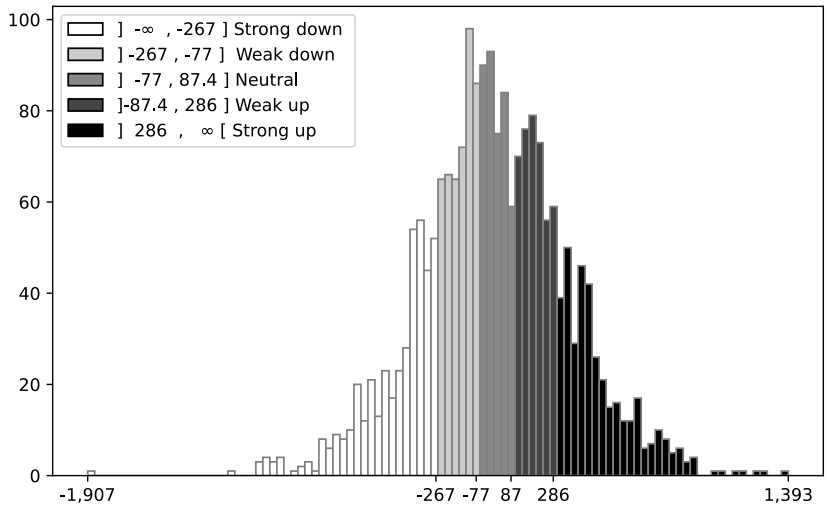


Figure 3.17: Example of the histogram for the qualitative classification of the output representing the integral of tick variation for the last 2 minutes: strong down (left white), weak down (light gray), neutral (gray), weak up (dark gray), and strong up (right black), respectively.

an example of the definition of target and stop-loss prices for each class of a given category.

Class	Mean of the ticks variation	Target	Stop-loss
Strong Up	6.44794	6	4
Weak Up	3.51428	4	3
Weak Down	-3.19424	3	2
Strong Down	-6.33173	6	4

Table 3.5: Example of trading mechanism parameters for a particular category.

Once the developed DL NN models are ready to go into production, they are subject to the final validation in the simulator described in Section 3.4.1 and Gonçalves et al. [2013].[3] To provide a better intuition of a trading execution based on the model prediction, Table 3.6 presents the log results of one trading execution. In the Appendix A2 is shown a stretch of the .csv output file containing fields of trading mechanisms parametrization and results, according to the DL models prediction during several races.

[3]Source codes are freely available online and can be consulted in this GitHub link: https://github.com/rjpg.

Instantiation	
Runner category	$nofavorite/mediumRunners/midleOdd/highLiquidit$ (Model #41)
Model predicted probabilities	[0.14 , 0.19, 0.17 , 0.20 , **0.30**]
Predicted class	5^{th} class: Strong Up
Bets/trade direction	Up: Lay (open) ⇒ Back (close)
Trading mechanism	Trailing Stop (strong movement predicted)
Parameters (in ticks)	Stop-loss: 4 , Target: 6
Parameters (in odds)	Entry odd: 4.6 , Target odd: 5.2 , Stop odd: 4.2
Time parameters	20 frames open, 80 frames start close best price, 20 close emergency
Open amount stake	£3.00 (Lay)
Potential PL	Profit: £0.35 , Loss: $-$£0.28
Result	
Trade final state	CLOSED
Moved ticks	6
Open amount stake	£3.00 (Lay)
Effective open odd (price)	4.6
Close amount stake	£2.65 (Back)
Effective PL	£0.35
Effective close odd (price)	5.2

Table 3.6: Example of one trading execution log given the category parameters and the model prediction.

Summary

This Chapter has a focus on the case studies used to test the developed MTS DL approaches empirically. They are UCI HAR, Beijing $PM_{2.5}$ levels, household electric power consumption, and Betfair betting exchange horse racing markets. For each case study, some bibliography study is made, the personalized FE applied is described in detail, and preliminaries are presented for the next Chapter. For the case study related to exchange markets, the end-to-end software framework project is detailed. The next Chapter focuses on the modeling stage.

Chapter 4

Methodologies and Results

In this chapter, we will explain in detail the DL methodologies that are the basis for the contributions of this thesis. Their application to a number of case studies will be illustrated with results. Let us explain in detail the base DL NN architectures considered in this study and the proposed extensions.

4.1 Models Architectures

4.1.1 CNN LeNet Based Models

LeCun et al. [1998a] proposed a NN architecture for handwritten and machine-printed character recognition which they called LeNet. The architecture, is based on convolution layers and is a type of straightforward CNN, simple to understand. The LeNet-5 architecture consists of 2 sets of convolutional and average pooling layers, followed by a flattening operation, then 3 dense layers (see Figure 4.1).

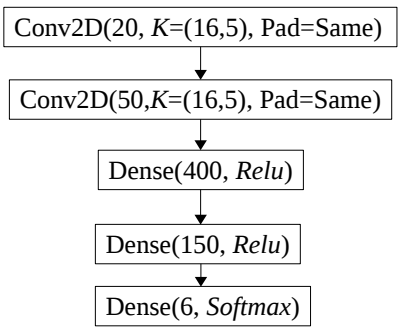


Figure 4.1: CNN 2D using same padding in convolutional layers

CNNs are also described in detail in LeCun et al. [2010]. This simple model is introduced here as a mere formalization and should be interpreted as a basic benchmark point relative to the alternative architectures. It is also used to identify improvements with the add-ons described through out this Section.

Consider a bi-dimensional input feature map x^l in the layer l of size (H, W) and a stride δ of $(1, 1)$. The simple mathematics for the computation of a convolutional layer l to obtain the output feature map y^l with kernel K of size (k_H, k_W) can be expressed as:

$$y^l_{i,j} = \phi^l(\sum_{i=0}^{H-k_H} \sum_{j=0}^{W-k_W} K \cdot {x^l}_{i,j}) \tag{4.1}$$

where ϕ is the activation function. This equation represents the application of kernel K in the input map x^l in layer l at coordinates i, j. A bias term b is usually added to $y^l_{i,j}$, which is omitted for a clearer presentation. An example of this computation, without applying the activation function, is given in Section 2.1.4, considering the convolution of input size 4×4 the 3×3 kernel K and stride $\delta = [1, 1]$, generating one output feature map of size 2×2. One observes that the output is smaller than the input when the convolution kernel is larger than $(1, 1)$. If the input has size (H, W) and the kernel $K = (k_H, k_W)$, then the convolution result has size $(H - k_H + 1, W - k_W + 1)$, which is smaller than the original input. Usually, this is not a concern for inputs with large dimensions (i.e., images) and small filters. However, it can constitute a problem with small input dimensions or when considering a high number of staked convolutional layers. As such, the practical effect of large filter sizes and/or very deep CNNs on the size of the resulting feature map entails loss of information such that the model can simply run out of data upon which it operates. The padding operation is conceived to tackle this issue.

4.1.1.1 Traditional Paddings

Currently, the standard procedure to avoid the border effect problem consists of applying same padding (i.e., the inclusion of zeros outside of the input map). For every channel of the bi-dimensional input x, we insert zeros $\frac{k_H-1}{2}$ rows above the first row and $\frac{k_H}{2}$ rows below the last row, and $\frac{k_W-1}{2}$ columns to the left of the first column and $\frac{k_W}{2}$ columns to the right of the last column. In this way, the convolution output size will be (H, W), thus, having the same spatial extent as the input. Note, however, that if the goal of the research is to analyze a MTS problem, the input feature map has a relatively small size in the variables component. Therefore, the inclusion of zeros through the same padding approach implies a weaker learning capability since the learned kernel is affected by the dot product operation with the included zero values, thus, potentially promoting an erroneous generalization.

	Example of padded info		
Method	Pad	Input	Pad
Valid (None)		a b c d e f	
Same (Zero)	0 0 0 0	a b c d e f	0 0 0 0
Reflect (Mirror)	d c b a	a b c d e f	f e d c
Reflect101	e d c b	a b c d e f	e d c b
Constant n	n n n n	a b c d e f	n n n n
Tile 2	a b a b	a b c d e f	e f e f
Causal (Zero Left)	0 0 0 0	a b c d e f	
Wrap	c d e f	a b c d e f	a b c d

Table 4.1: Padding examples of size 4 for unidimensional input.

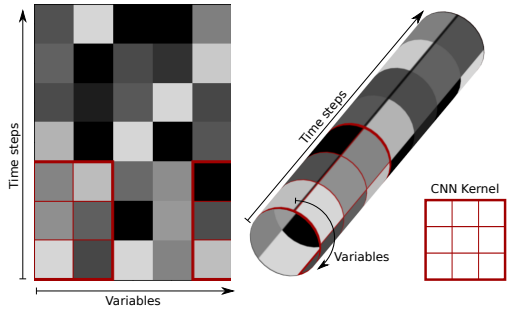
According to Hamey [2015], there are other known padding methods that are commonly provided in image processing environments. These make use of the information in the input x to fill in the borders. Table 4.1 exemplifies each one. One contribution of this dissertation consists in providing a new type of padding to the existing literature.

4.1.1.2 Roll Padding

Roll padding is an extension of wrap padding conceived for MTS analysis. Wrap copies information from the opposite sides of the image, effectively mapping the image onto a torus. This operation corresponds to four copies of information under a bi-dimensional input. The wrapped rows (columns) above the top (on the left) of the input are a copy of the bottom rows (right columns), respectively and vice-versa. Although wrapping is not typically useful for natural images, it is very appropriate for computed images such as Fourier transforms and polar coordinate transforms where pixels in opposite borders are computationally adjacent.

As observed in Figure 4.2, roll padding copies information from the opposite sides but only in one dimension, which is the variables component in the MTS bi-dimensional input map (i.e., *TimeSteps* $\times$ *Variables*). In turn, the time steps component remains with no padding (i.e., valid) resulting in a cylinder instead of a torus. The reduction of time steps component after several convolutions is not problematic due to the frequent presence of a high number of time steps in this type of problems. Nevertheless, for the time steps component, roll padding can be combined with other types of padding methods (e.g., causal) as exemplified in Figure 4.10 and Subsection 4.1.4.

As clarified in Figure 4.3, the skeleton of CNNs using roll padding is also derived from LeNet-5 [Lecun et al., 1998]. Both CNNs use exactly the same hyperparameters, the only difference relies on the type of padding employed. In this way, we can infer implications of the use



	Roll	Variables	Roll
Valid			
Time	cdef ijkl opqr	abcdef ghijkl mnopqr	abcd ghij mnop
Valid			

Figure 4.2: Roll padding scheme in MTS analysis.

of roll padding at the performance level. Note that, in Figure 4.3 for the second CNN, we specify the padding method employed in each dimension. The valid padding is used for the time steps component, while the roll padding is considered for the variables component.

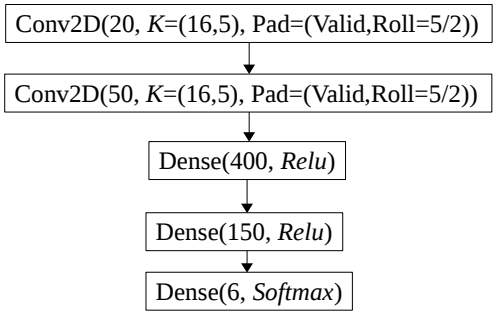


Figure 4.3: CNN 2D using valid padding in time steps component and roll pading, of size $\frac{K_H}{2}$, in the variables component.

4.1.2 LSTM Based Models

The base LSTM [Hochreiter and Schmidhuber, 1997] employed in this study is constituted by four layers, three of them using bidirectional LSTMs plus a dense with softmax for classification (see Figure 4.4). Likewise in Section 4.1.1 this model serves as a base point to verify improvements when docking extra methodologies to it. RNN and LSTM inner-working is explained in Section 2.1.2. RNNs have been used successfully for many tasks involving sequential data. Improved RNN models, such as LSTMs, enable training on long sequences overcoming problems like vanishing gradients. However, even the more advanced models have their limitations and researchers had a hard time developing high-quality models when working with long data sequences. Many MTS problems have to find connections between long input and output, between layers, composed of dozens of time steps. The existing RNN

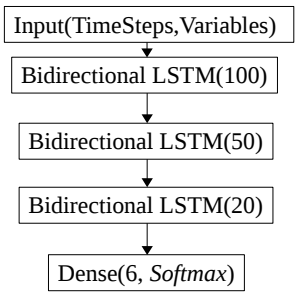


Figure 4.4: Stacked Bidirectional LSTMs

architectures needed to be changed and adapted to better deal with such tasks.

4.1.2.1 Standard Attention

Attention is a mechanism to be combined with RNN allowing it to focus on certain parts of the input sequence when predicting a certain part of the output sequence, enabling a faster and resilient convergence. Combination of attention mechanisms enabled improved performance in many tasks making it an integral part of modern RNN networks. A very important paper about attention is Vaswani et al. [2017]. It was originally introduced for machine translation tasks, but it has spread into many other application areas. On its basis attention can be seen as a residual block that multiplies the result with its own input h_i and then reconnects to the main NN pipeline with a weighted scaled sequence. This scaling parameters are called attention weights α_i and the result is called context weights c_i for each value i of the sequence, all together, are called context vector c of sequence size n. This operation is given by :

$$c_i = \sum_{i=0}^{n} \alpha_i h_i \tag{4.2}$$

Computation of α_i is given by applying a softmax activation function to the input sequence x^l on layer l:

$$\alpha_i = \frac{\exp(x_i^l)}{\sum_{k}^{n} \exp(x_k^l)} \tag{4.3}$$

Meaning that the input values of the sequence will compete with each other to receive attention, knowing that, the sum of all values obtained from the softmax activation is 1, the scaling values in the attention vector α will have values between $[0, 1]$. The mechanism we described previously is called *soft attention* because it is a fully differentiable deterministic mechanism that can be plugged directly into a backpropagation based system. The gradients

are propagated through the attention block the same way they are propagated through the rest of the network. *Hard attention*, instead of a weighted average, uses α_i as a sample rate to choose if x_i will be considered in the context vector. *Hard attention* replaces a deterministic method with a stochastic sampling model. To calculate the gradient descent correctly in the backpropagation for the entire NN, inside the *hard attention* block samplings are performed using the Monte Carlo method, and then results are averaged. Monte Carlo performs end-to-end episodes to compute an average for all sampling results [Xu et al., 2015]. The accuracy is subject to how many samplings are performed and how well it is sampled. On the other hand, *Soft attention* follows the regular and easier application of backpropagation, in the computation of the gradients inside the attention block. However, the accuracy is subject to the assumption that the weighted average is a good representation for the area of attention. Both have their shortcomings. Currently, *Soft attention* is more popular because the seamless and direct backpropagation application seems more effective. For this dissertation we only use *soft attention*.

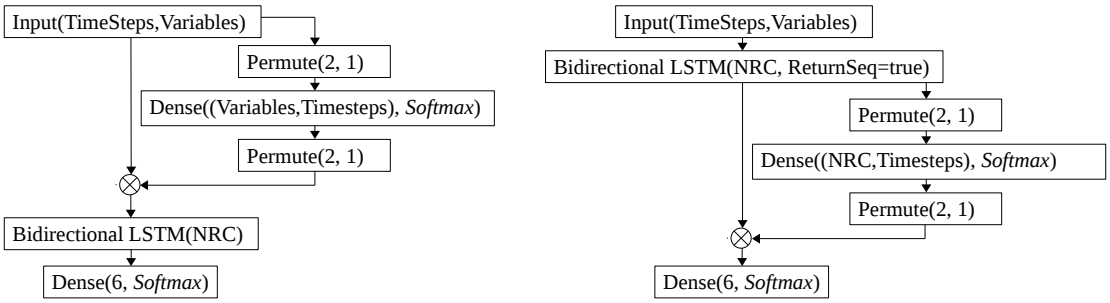


Figure 4.5: MTS attention before LSTMs on the left subplot and attention after LSTM on the right subplot.

If attention is applied to the input directly, before enter the LSTM (or any type of recursive layer), we call it *attention before*, otherwise, if it is applied to the LSTM output sequence, it is called *attention after* as represented in Figure 4.5. Since we are dealing with MTS we use a bi-dimensional dense layer for attention and therefore we permute before and after this layer so the attention mechanism is applied in the time steps component of each sequence and not in the variables component. It is important to highlight that when the attention is applied after, the LSTM layer must return the internal recursive generated sequences, which is equal to the number of units defined NRC. This parameter is used inside the attention block to know how many sequences it must process.

4.1.2.2 Multi-Head Convolutional Attention

A contribution of this dissertation is the introduction of convolutional layers inside the attention block. Attention primordial design was made for text processing in long sequences of words. For each embedded word in the sequence, a level of attention is given individually. For a "less discrete" type of problem involving time series, it can be useful to give attention to patterns in small contiguous segments instead of individual values.

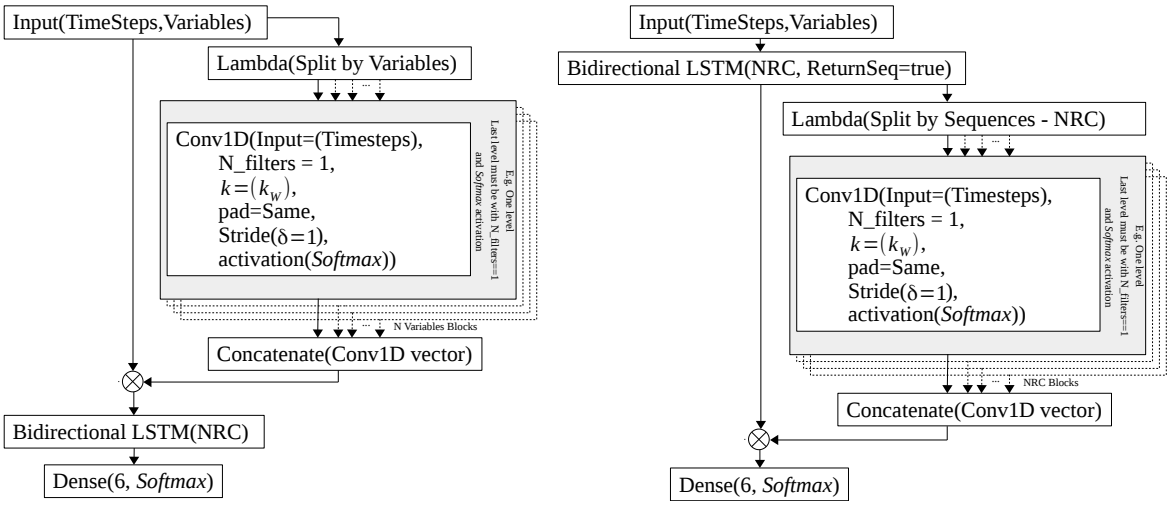


Figure 4.6: Attention using convolutional layers before and after LSTMs.

In Figure 4.6 is illustrated the implementation process. The MTS is split into individual time series, using Keras Lambda function. For each sequence is created a path with 1D convolutional layers and the result is concatenated again. In Figure 4.6 is illustrated only one filter convolution per sequence, i.e., per variable of the MTS, if attention before LSTM, or per Number of Recursive Cell (NRC) generated sequence, if after LSTM. It is important that before the concatenation operation each path return a one dimensional vector with size *TimeSteps*. This vector concatenated with the others results in a attention weights feature map of size $TimeSteps \times Variables$. This map is compatible for multiplication with h to obtain the 2D context map c. To have many filters, i.e., to process many small sub sequence patterns, we must stack multichannel 1D convolution layers before. However, the last one, inside the attention block, must return only one channel, as explain before. Another way to force a 1D output vector for each path would be using the *AveragePooling1D* keras layer to average previous channels into one dimension. Also, this last single channel 1D convolution output, must use the softmax activation so each value, in the resulting vector per variable, competes with each other, summing to 1, and has a scaling factor in $[0, 1]$ range.

4.1.3 ConvLSTM2D Based Models

4.1.3.1 ConvLSTM2D for Segmented Time Series

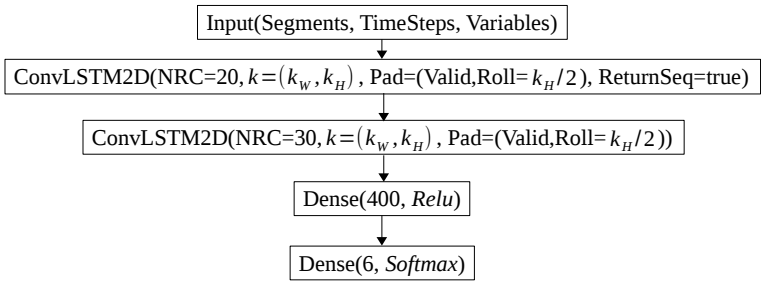


Figure 4.7: Base scheme for staked ConvLSTM2D with roll padding on the variables component.

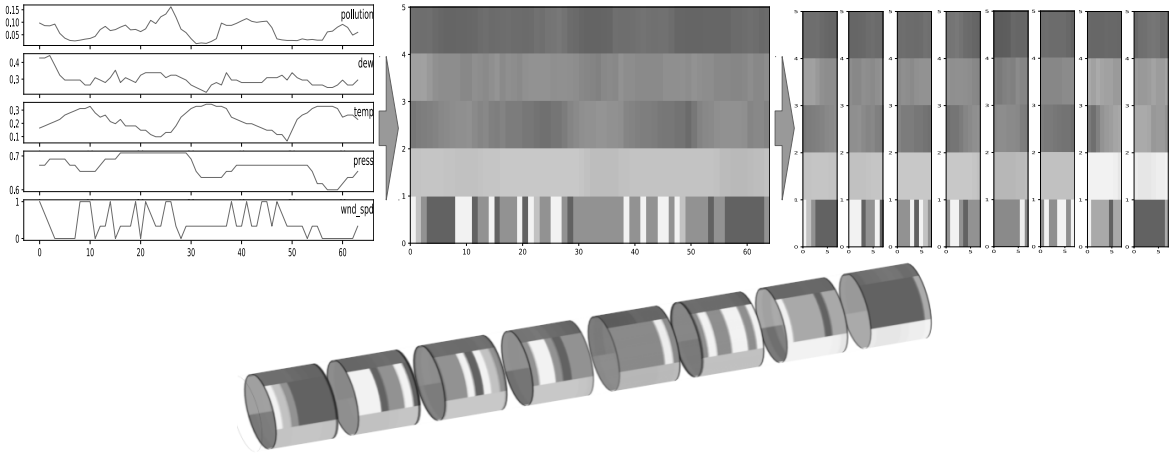


Figure 4.8: MTS input processing for ConvLSTM2D. The bottom subplot describes the application of roll padding in the variables component for each segment.

The ConvLSTM2D layer was proposed by Shi et al. [2015]. The motivation of this structure was to predict future rainfall intensity based on sequences of meteorological images. Applying this layers in a NN architecture they were able to outperform state-of-the-art algorithms for this task. The ConvLSTM2D is a recurrent layer, just like the LSTM, but internal matrix multiplications are exchanged with convolution operations. As a result, the data that flows through the ConvLSTM2D cells keeping the input dimension, 3D in our case, *Segments* $\times$ *TimeSteps* $\times$ *Variables*, instead of being just a 2D map, *TimeSteps* $\times$ *Variables* (see Figure 4.8). As explained in Section 4.1.1.2 we can also apply roll padding in this input on the variables component. ConvLSTM2D layers can be useful in MTS that can be partitioned into segments, e.g., the household electric power consumption case study (see section 3.3).

The time series will have representative patterns for every day of the weak that can be grouped and contained in a 2D map.

4.1.3.2 ConvLSTM2D Convolutional Attention with Roll Padding

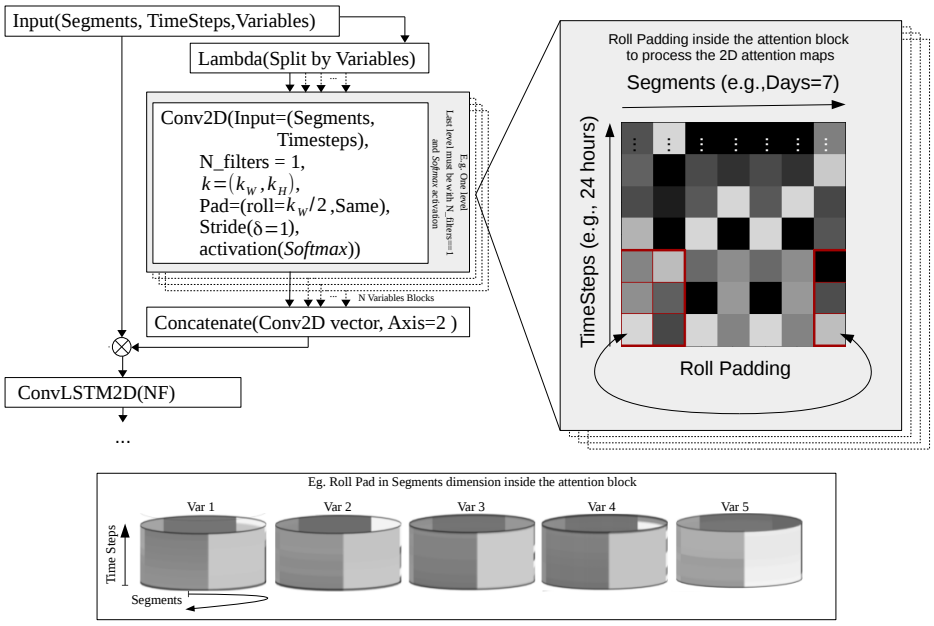


Figure 4.9: Attention using 2D convolutional layers before ConvLSTM2D. The bottom subplot describes the application of roll padding in the segments component for each variable inside the attention block.

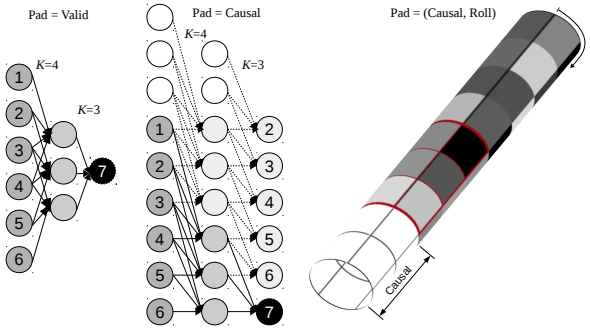
When entering the attention block, after splitting by variables, the resulting 2D map, to be processed by convolution layers, will have *Segments* × *TimeSteps* format. This means the 2D kernels will try to capture patterns relating contiguous time steps, and the same temporal steps in the previous and next segments. E.g., if the segments represent days and the time steps are divided by hours a 2D kernel will capture attention patterns relating some hours of the day and also the same period in the days before and after. Moreover, if we have segments of 7 days we can use roll padding in the segments component so the border processing, by the kernel, can correlate the first day of the week with the last day of the week if the data tends to have a strong weekly cycle (see Figure 4.9). This technique is useful in MTS that exhibit cyclic properties. If it is not desirable to correlate data between segments an one dimension kernel must be defined (i.e., 2D $K = (1, k_w)$ since we are in a bi-dimensional convolution layer). Each 2D output map will be the result of a sofmax activation. Each value, in the resulting 2D map per variable, competes with each other, summing all to 1, with a scaling factor between $[0, 1]$, as explain before. After all convolutions/2D maps are concatenated,

the resulting α will be 3D and compatible to scale the inputs h of the attention block to obtain c, as described in equation 4.2.

4.1.4 Multivariate WaveNet

4.1.4.1 WaveNet 1D with Multichannel Input

One relevant DL architecture applied in time series is WaveNet from Google DeepMind [van den Oord et al., 2016]. WaveNet was originally developed for audio signal generation. One important component developed to accomplish this task was a sound classifier. The WaveNet sound classifier is based on 1D convolutional layers. For unidimensional data the causal convolution is implemented by shifting the input a few time steps behind as described in Section 2.1.6. In the first two left subplots of Figure 4.10, one can observe the output difference between resorting or not to the use of causal padding in 1D convolutions. Causal



	Roll	Variables	Roll
Valid			
Time	cdef ijkl opqr	abcdef ghijkl mnopqr	abcd ghij mnop
Causal	0000 0000	000000 000000	0000 0000

Figure 4.10: On the left subplot, comparison behavior between valid and causal padding. On the right subplot, combination of causal and roll padding scheme for MTS analysis with WaveNet 2D.

padding allows to preserve the time steps of past information for the next layer. In MTS this requires that the number of zeros to be added before the beginning of all sequences is given by $k-1$, being k the size of the unidimensional kernel for 1D convolutions. Note that, for MTS inputs, each variable is treated as a channel in 1D convolutions and the causal padding is applied equally to every channel.

WaveNet uses dilated convolutions to gradually increase the receptive field. Thus, in the time steps component, when using dilation rate dr (i.e., for $dr > 1$) the causal padding has a size given by $dr \times (k-1)$. The residual block of the WaveNet architecture is executed a given number of times in $depth$ in the network, with $N = \{1, ..., depth\}$. Inside this block, the dilatation dr of *sigmoid* and *Tanh* convolutions, applied to the time steps component,

increases exponentially according to the formula $dr = k^N$. The third and final convolution inside the residual block (see Figure 4.11 and 2.16) has $k = 1$ and $dr = 1$ for dimensionality reduction and manage model complexity. This layer is also called *channel-wise pooling layer*.

The standard WaveNet uses 1D convolutions and we can adapt to MTS problems by simply using the input as a multichannel of several 1D sequences of variables. In this thesis, to test the inclusion of roll padding in the variables component, the WaveNet is extended to work with 2D convolutions.

4.1.4.2 WaveNet Extended with 2D Convolutions and Roll Padding

Figure 4.11 shows the extended WaveNet to work with 2D maps instead of 1D multichannel. The developed architecture maintains the standard WaveNet processing in the time steps component, using causal padding, while it introduces roll padding in the variables component. Since we are using 2D convolutions now, the kernel size is defined by (k_H, k_W). The k_H component (i.e., time steps component) is processed as explained in Section 4.1.4.1, for k. Also in this time steps component, the dr_H and causal padding is applied as described in Section 4.1.4.1. The combination of both types of padding, causal and roll, is clarified in the last two right subplots of Figure 4.10. In the second dimension (i.e., variables component) the use of roll padding is illustrated, whose size depends on k_W. We establish a roll padding of size $\frac{k_W}{2}$, copying opposite $\frac{k_W}{2}$ columns information of the input map. For simplicity, we assume odd fixed sizes in k_W. No dilation rate is considered in the variables component (i.e., $dr_W = 1$).

In short, everything is processed likewise the basic WaveNet philosophy in the time steps component, while everything is processed as normal convolutional layers with roll padding in the variables component. Finally, after adding the skip connections, three 2D convolutional layers are considered. In this scheme, we use a stride $\delta = (\delta_H, \delta_W)$ with $\delta_H > 1$ and $\delta_W = 1$ to down sample only the time steps dimension rather than considering pooling layers. The last convolution has x filters ($x = 6$ in Figure 4.11) to generate x feature maps in which a global average pooling is applied. In this way, we use softmax directly in the x resulting values for classification of x number of classes.

4.2 Results

This Section presents the results obtained for the case studies described in Chapter 3. Some case studies, due to their nature, have relatively low accuracies. Nevertheless, it is shown that small gains can have a significant impact on some problems with low precision ranges. The

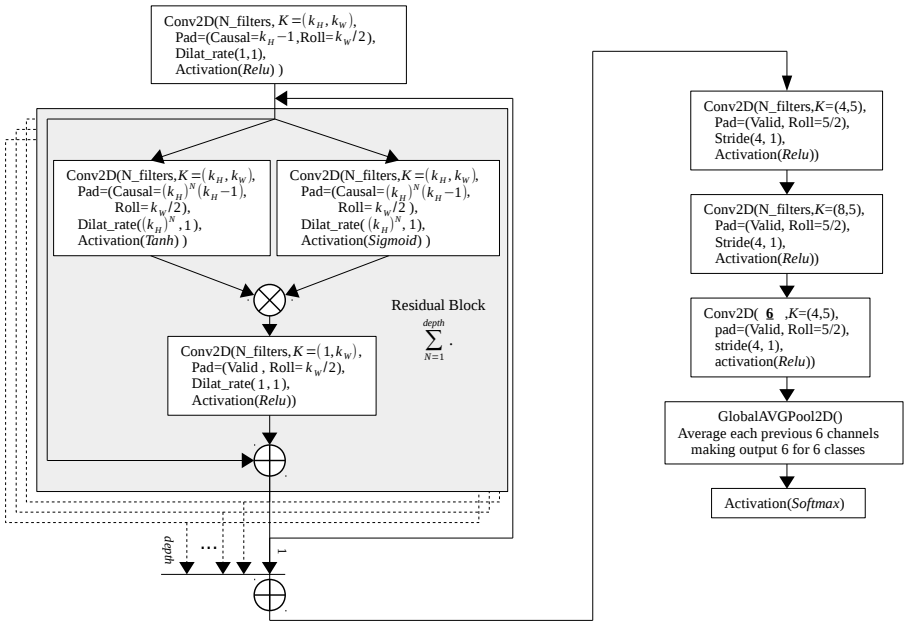


Figure 4.11: WaveNet 2D architecture for MTS classification using 2D convolutions with causal padding in the time steps component and roll padding in the variables component.

main goal is to compare models and study the influence of the add-ons described throughout Section 4.1 of this Chapter.

4.2.1 HAR - UCI Dataset

This Section highlights the results related to the HAR dataset (see Section 3.1). Comparative results are mainly related to applying standard LSTM, CNN with and without roll padding, and the best model of all WaveNet 2D with roll padding. With these models, we can infer about the introduction of roll padding.

Figure 4.12 presents the evolution of accuracy and loss in the train and test datasets and the epoch threshold. The highest accuracy is achieved, which corresponds to the point where the best model is found. These charts result from the best run, in 5 repetitions, of a learning process composed of 150 epochs. 150 epochs were not needed for some models, resulting in overfitting, but are presented for global comparison. Observing the loss's variance, it is also clear that the partition for the validation set should be increased. Note that this case study has a standard partitioning established by the data providers [Anguita et al., 2013].

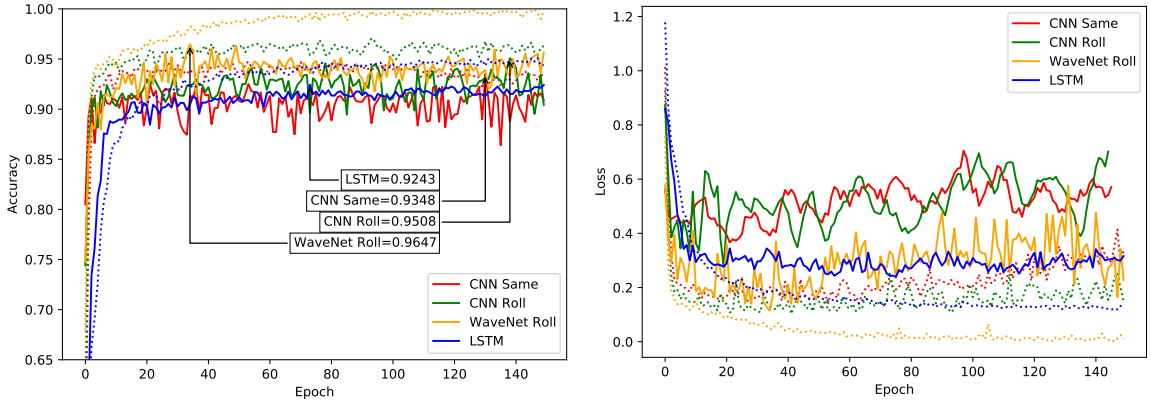


Figure 4.12: Learning process evolution. Accuracy on the left subplot and loss (i.e., categorical cross-entropy) on the right subplot. Doted lines refer to the train dataset, while solid lines refer to the test dataset.

In terms of accuracy, one observes that the CNN with same padding has a similar evolution to the LSTM. On average, the CNN with roll padding has a slightly superior accuracy relative to the previous models both in the train and test datasets. Therefore, the first relevant conclusion is the improvement of accuracy with the introduction of roll padding in the CNN. Moreover, the WaveNet 2D model with roll padding surpasses all the alternative options in terms of accuracy. As such, the second relevant conclusion is that the incorporation of roll padding in the WaveNet allows to achieve the best performance result.

In terms of loss, the WaveNet 2D model with roll padding is the best model in the train and test datasets. At the training level, loss values are very close to zero. The associated accuracy reaches near 100% in some epochs, which seems to suggest there is some room for improvement in the test dataset results by choosing the right amount of regularization.

The second best loss is achieved by the LSTM. Furthermore, CNN with same padding and CNN with roll padding have equivalent loss values. A lower loss in the LSTM and a higher accuracy in the CNN with roll padding means that the later model responds with great certainty to some wrong guesses, which drastically penalizes its error but such effect is not passed onto the level of accuracy. Convolutional based models tend to have more sparsified outputs in relation to LSTMs due to the heavy use of ReLU activation function. At this point, it is important to highlight that the error function is the categorical cross-entropy. The observed phenomenon is relatively common in the development of DL models given that, from a practical point of view, neglecting a small increment of the loss can sometimes allow to obtain benefits at the accuracy level.

Table 4.2 summarizes results by class of the best DL model : WaveNet 2D. This corresponds

		Predicted							
	Classes	Walking	Upstairs	Downstairs	Sitting	Standing	Laying	Recall(%)	ACC (%)
Real	Walking	474	15	4	1	1	0	95.77	**96.47**
	Upstairs	2	462	0	5	2	0	98.09	
	Downstairs	2	3	415	0	0	0	98.81	
	Sitting	0	0	0	458	33	0	93.28	
	Standing	1	0	0	33	498	0	93.61	
	Laying	0	2	0	0	0	535	99.63	
	Precision (%)	98.96	95.85	99.05	92.15	93.26	100		

Table 4.2: Confusion matrix for the best run applying the WaveNet 2D with roll padding model on the HAR RTD validation dataset as provided by UCI.

to the point where the final best model is obtained for the case study described in Section 3.1. Table 4.3 presents accuracies related to all models described in Section 4.1 and respective baselines. The ConvLSTM2D-based model was configured with 8 segments and 16 time steps for each segment (see Section 4.1.3). This case study and the obtained results are also described in Gonçalves et al. [2021]

Model	Min	Max	Mean	Variance
CNN	0.9226	0.9348	0.9262	2.4503e-05
CNN ROLL	0.9423	0.9508	0.9460	1.0143e-05
LSTM	0.9219	0.9243	0.9231	1.0132e-06
LSTM Att.	0.8571	0.8646	0.8609	1.1802e-05
LSTM Att. Conv1D	0.8629	0.8781	0.8692	3.4980e-05
ConvLSTM2D	0.9127	0.9141	0.9133	3.7997e-07
ConvLSTM2D Att. Conv2D	0.9298	0.9321	0.9309	1.0132e-06
WaveNet	0.9545	0.9596	0.9575	3.5349e-06
WaveNet2D ROLL	0.9579	**0.9647**	**0.9612**	7.6225e-06

Table 4.3: Descriptive statistics of accuracies obtained with 5 repetitive runs of the fitting process for each model applied to the UCI HAR case study.

4.2.2 Air Pollution - $PM_{2.5}$ Concentration

This Section presents the results related to the classification of $PM_{2.5}$ concentration in the air as described in Section 3.2. Table 4.4 reveals that the standard ConvLSTM2D has the

Model	Min	Max	Mean	Variance
CNN	0.3969	0.4080	0.3997	2.2307e-05
CNN ROLL	0.4122	0.4226	0.4163	1.6030e-05
LSTM	0.4121	0.4208	0.4150	1.3006e-05
LSTM Att.	0.4035	0.4139	0.4080	1.6030e-05
LSTM Att. Conv1D	0.4035	0.4122	0.4066	1.4215e-05
ConvLSTM2D	0.4243	**0.4348**	0.4285	1.9055e-05
ConvLSTM2D Att. Conv2D	0.4157	0.4226	0.4191	1.2098e-05
WaveNet	0.3983	0.4173	0.4062	6.1399e-05
WaveNet2D ROLL	0.4253	0.4305	0.4281	3.9183e-06

Table 4.4: Descriptive statistics of accuracies obtained with 5 repetitive runs of the fitting process for each model applied to the air pollution case study.

highest accuracy. However, we also observe that the application of roll padding improved the accuracy of CNN-based models and WaveNet-based models. We can also conclude that attention mechanisms did not have a positive effect on accuracy. The ConvLSTM2D-based model was configured with 6 segments and 12 time steps for each segment (see Section 4.1.3). This means that the LSTM part receives segments of size 6, where each is composed of 12 hours processed in convolution.

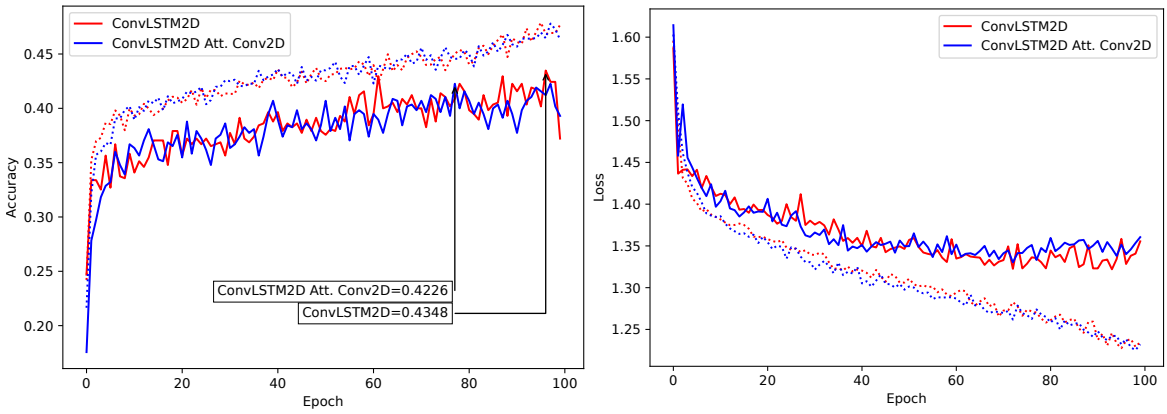


Figure 4.13: Learning process evolution of the ConvLSTM2D-based models for the air pollution case study. Accuracy on the left subplot and loss (i.e., categorical cross-entropy) on the right subplot. Doted lines refer to the train dataset, while solid lines refer to the test dataset.

Figure 4.13 compares the learning curves between the models of the ConvLSTM2D family. Both are similar, suggesting that a correct parametrization of the input time windows for an optimal processement of the developed attention mechanisms can further improve

		Predicted						
	Classes	Level 1	Level 2	Level 3	Level 4	Level 5	Recall(%)	ACC (%)
Real	Level 1	85	20	5	2	3	73.91	**43.48**
	Level 2	37	63	24	9	9	44.37	
	Level 3	14	48	46	17	15	32.86	
	Level 4	5	25	32	19	27	17.59	
	Level 5	5	9	15	4	37	52.86	
	Precision (%)	58.22	38.18	37.70	37.25	40.66		

Table 4.5: Confusion matrix for the ConvLSTM2D model without add-ons. Best model on the air pollution validation dataset.

the accuracy. Note that the developed attention mechanism with roll padding applied to ConvLSTM2D-based models relies on the assumption of cyclicity (e.g., weekly) in the input sequence. This issue was addressed in the household electric power consumption case study further analyzed. Table 4.5 presents the confusion matrix of the best DL model.

4.2.3 Household Electric Power Consumption

This Section highlights the results related to the household electric power consumption dataset (see Section 3.3). Table 4.6 presents the main results, which are summarized as

Model	Min	Max	Mean	Variance
CNN	0.3916	0.4090	0.3986	4.2789e-05
CNN ROLL	0.4196	0.4301	0.4224	2.0783e-05
LSTM	0.4266	0.4336	0.4308	8.5578e-06
LSTM Att.	0.3986	0.4126	0.4055	4.8902e-05
LSTM Att. Conv1D	0.4231	0.4336	0.4266	2.4451e-05
ConvLSTM2D	0.4231	0.4371	0.4273	3.3009e-05
ConvLSTM2D Att. Conv2D	0.4372	**0.4476**	**0.4413**	2.0783e-05
WaveNet	0.4021	0.4336	0.4147	1.5648e-04
WaveNet2D ROLL	0.4231	0.4440	0.4308	7.5798e-05

Table 4.6: Descriptive statistics of accuracies obtained with 5 repetitive runs of the fitting process for each model applied to household electric power consumption case study.

follows. First, we find evidence that introducing roll padding in a standard CNN improves

the accuracy level for this MTS problem. Moreover, we conclude that top 3 best models are the ConvLSTM2D using multi-head 2D convolution attention with roll padding, WaveNet2D with roll padding, and standard ConvLSTM2D. For the 3D tensor input of ConvLSTM2D-based models, we use *7 Segments* × *24 TimeSteps* × *7 Variables*. This means convolutions will extract features for each day and the LSTM architecture will process these features over sequences of 7 days. Moreover, this produces the ideal setup to apply the multi-head convolutional 2D attention with roll padding as described in Section 4.1.3.2.

Table 4.7 presents the confusion matrix of the best DL model in classification mode. Results

		Predicted						
	Classes	Level 1	Level 2	Level 3	Level 4	Level 5.	Recall(%)	ACC (%)
Real	Level 1	31	14	8	0	2	56.36	**44.76**
	Level 2	10	53	7	0	1	74.65	
	Level 3	0	38	23	6	6	31.51	
	Level 4	0	25	14	15	5	25.42	
	Level 5	1	9	7	5	6	21.43	
	Precision (%)	73.81	38.13	38.98	57.69	30.00		

Table 4.7: Confusion matrix for the ConvLSTM2D-based model with roll padding, on the variables component, and multi-head (per variable) attention with roll padding, on the 7 segments (days of the weak), inside the attention block. Model execution on the household electric power consumption test dataset, i.e., last 20% of global data.

indicate that the best DL model outperforms the ARIMAX. Although the accuracy of 44.76% may seem a low value at first glance, we must consider that are dealing with a multi-classification problem in which the dependent variable is segmented into 5 classes. Knowing that these 5 classes correspond to a ranking of the output, the most important aspect is to observe to which extend the values of the confusion matrix approximate to the diagonal. Note that the accuracy corresponds only to the values that fall precisely into the diagonal.

Model	MSE	RMSE	MAE
ARIMAX	0.01648	0.12838	0.09259
ConvLSTM2D Att. Conv2D	**0.01400**	**0.11831**	**0.08746**

Table 4.8: Forecasting error metrics of the two models in the test dataset for the normalized output.

For this case study, we compare the best DL model with the classical ARIMAX assuming

daily observations as the time unit (see Section 2.3). To perform the analysis, we change the output layer of our model to work in regression mode. In particular, we change the 5 output neurons with softmax activation to a single neuron with tanh activation, since the output is normalized between [-1,1], and the error function from categorical cross-entropy to MSE. To test whether the regression problem provides better insights than the classification mode, we create a confusion matrix based on the dependent variable's predicted values resulting from the regression analysis plugged into the predefined intervals. It is a common strategy observed in Kaggle competitions for multi classification ordered problems. As a result from applying this technique, the accuracy increases. This result is expected since the categorical cross-entropy does not take into consideration that classes are ordered. Note that the goal in this Section is not to achieve the best possible refined output value for the DL model, but rather to compare models maintaining the same structure. Nevertheless, we optimize hyperparameters for the ARIMAX model with a grid search, whose optimal result gives $(p, d, q) = (2, 0, 1)$ (see Section 2.3). Figure 4.14 shows the regression output on the test

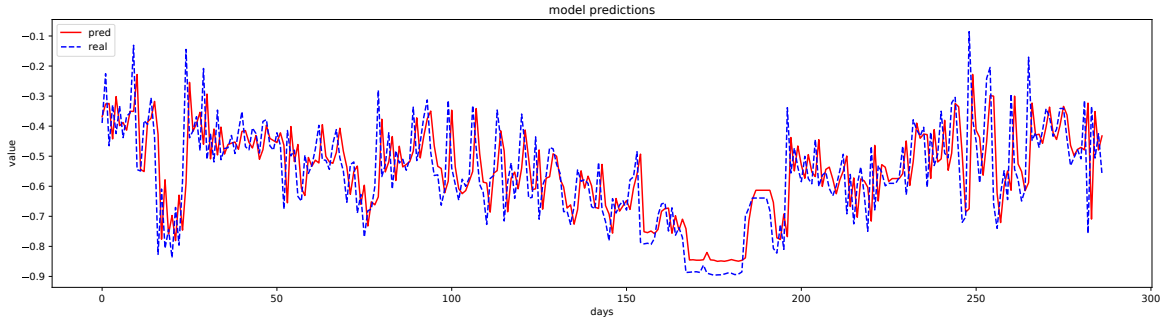


Figure 4.14: Step by step regression outputs vs real values for the ConvLSTM2D with Conv2D attention mechanism and roll padding. The test set corresponds to the last 286 days.

dataset and Table 4.9 presents the confusion matrix of the output of the model, in regression mode, converted into classes.

4.2.4 Betting Exchange - Horse Racing Markets

This Section focus on the results related to the betting exchange markets (see Section 3.4). An end-to-end analysis is performed for this case study since a complete framework was established, from data gathering to market interaction. Table 4.10 shows the best result is achieved using an LSTM-based model with multi-head attention processing each variable evolution with Conv1D. LSTM models outperform the other presented alternatives. As such, the intrinsic nature behind the LSTM philosophy seems to fit into this type of data adequately. It is also visible that the inclusion of roll padding in the simple CNN and

		Predicted						
	Classes	Level 1	Level 2	Level 3	Level 4	Level 5.	Recall(%)	ACC (%)
Real	Level 1	32	16	4	1	2	58.18	**48.95**
	Level 2	9	40	18	4	0	56.34	
	Level 3	2	20	35	9	7	47.95	
	Level 4	0	4	23	27	5	45.76	
	Level 5	1	5	10	6	6	21.43	
	Precision (%)	72.73	47.06	38.89	57.45	30.00		

Table 4.9: Confusion matrix for the ConvLSTM2D-based model in regression mode with 2D multi-head attention and roll padding. The results of the model in regression mode are converted back into the respective consumption level interval (see Figure 3.7).

WaveNet improves their base models accuracy. However, due to the relatively low sampled data in this case study, these CNN-based models tend to be very sensitive to overfitting and smaller CNNs based solutions with convergence should be further investigated.

Model	Min	Max	Mean	Variance
CNN	0.2342	0.2488	0.2381	3.6756e-05
CNN ROLL	0.2391	0.2536	0.2430	3.9674e-05
LSTM	0.2826	0.2874	0.2840	4.6675e-06
LSTM Att.	0.2681	0.2826	0.2720	3.9674e-05
LSTM Att. Conv1D	0.2971	**0.3092**	0.3005	2.5088e-05
ConvLSTM2D	0.2584	0.2801	0.2671	7.1763e-05
ConvLSTM2D Att. Conv2D	0.2608	0.2705	0.2642	1.3419e-05
WaveNet	0.2512	0.2681	0.2589	7.1180e-05
WaveNet2D ROLL	0.2826	0.2922	0.2864	1.3419e-05

Table 4.10: Descriptive statistics of accuracies obtained by 5 repetitive runs of the fitting process for each model applied to the betting exchange case study.

Table 4.11 presents the best DL model, which reaches 30.92% of accuracy. Although it only goes 11 p.p. above the baseline, given that we have 5 classes equally distributed, we need to consider that the goal of this problem is to obtain a positive PL. Some wrong predictions can still generate a positive PL, e.g., a predicted movement to the weak up class when the real movement is a strong up class. All cases with a positive PL are identified in Table 4.11 by the green color. Analogously, similar is applied to the values represented by the red

color, but with the difference that a negative PL is generated. The DL model's marginal convergence makes values approximate to the principal diagonal, making them attracted to the green cells and repelled from the red cells, thus allowing them to obtain a profitable model even with low accuracy. For this case in particular, knowing that 20% of data are used to validation, the total number of predicted trades with expected positive PL is equal to 173, the total number of predicted trades with expected negative PL is equal to 98, so that there are 75 expected positive predicted trades in total.

To truly understand its potentialities, we need to analyze the model in production given that numerous factors can affect the trade execution. This task resorts to a final test dataset of 30 days ahead, that has not been used in the modeling phase, to test the model in production.

		Predicted						
	Classes	Strong Down	Weak Down	Neutral	Weak Up	Strong Up	Recall(%)	ACC (%)
Real	Strong Down	32	7	11	9	10	46.38	**30.92**
	Weak Down	39	9	17	10	11	10.47	
	Neutral	27	6	23	9	18	27.71	
	Weak Up	23	13	19	9	27	9.89	
	Strong Up	14	8	15	10	38	44.71	
	Precision (%)	23.70	20.93	27.06	19.15	36.54		

Table 4.11: Confusion matrix for LSTM-based model with Conv1D multi-head attention on the validation dataset.

Figure 4.15 presents the cumulative PL of the trades' execution in simulation (see Section 3.4) in the final test dataset for one category. The left-hand side subplot presents the absolute PL obtained with stakes of £3.00 and £100.00, while the right-hand side subplot shows the relative PL with stakes of £3.00 and £100.00 concerning the initial investment. When executing the same predicted trades, one can observe that using stakes of £3.00 generates a higher ROI compared to stakes of £100.00, which is explained by the absorption capability of the market. Further studies should evaluate the optimal stake policy that maximizes the absolute PL.

Table 4.12 summarizes the number of executed trades, greens, reds, positive and negative ticks for the best DL model. The number of trades is the number of times one trading mechanism is instantiated. Greens is the number of times the trading mechanism closes

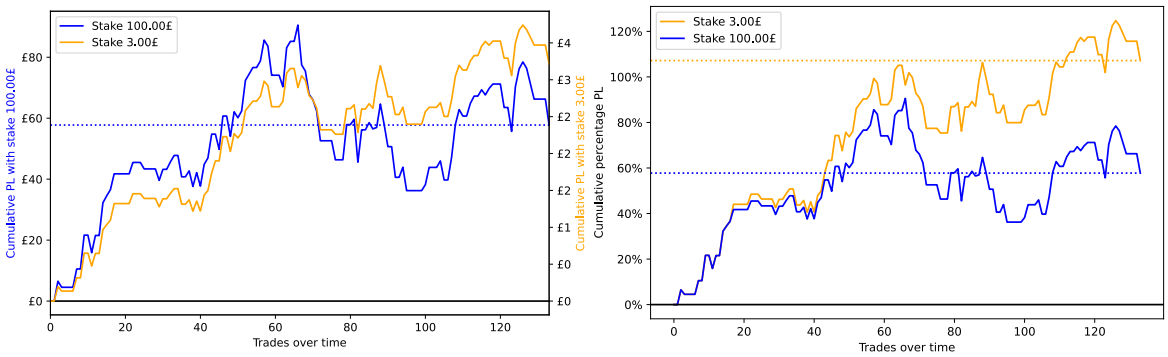


Figure 4.15: Evolution of the PL during 30 days of trading using the best model, of one category (#41), with stakes of £3.00 and £100.00. On the left subplot, absolute PL values. On the right subplot, relative PL values in relation to the investment stake.

Stake	£3.00	£100.00
Trades	134	134
Greens	54	50
Reaches target	14	13
Closes]null;target[	40	37
Swings	14	13
Trailing-Stops	40	37
Reds	36	39
Reaches stop-loss	13	14
Breaks stop-loss	3	5
Closes]null;stop-loss[	23	25
Swings	10	11
Trailing-Stops	26	28
Null	44	45
Positive ticks	134	129
Swings	31	28
Trailing-Stops	103	101
Negative ticks	87	100
Swings	23	26
Trailing-Stops	64	74

Table 4.12: Global trading simulation results with the model in production on the final test dataset.

in profit. Reds is the number of times the trading mechanism had to close in loss. The sum of greens and reds is not equal to the number of trades because sometimes the trading mechanism can close the trade on the same entry price without making a profit or a loss. This can happen when the trading mechanism reaches the timeout exposure and closes at the same entry price. For these cases the result is null. Also, when the opening bet is not matched during the opening time, the result is null. Positive ticks is the number of total ticks that result from profitable trades. Negative ticks is the total number of ticks that result in loss. These are the main values to get conclusions about how well-succeeded a trading policy is.

		Predicted						
	Classes	Strong Down	Weak Down	Neutral	Weak Up	Strong Up	Recall(%)	ACC (%)
Real	Strong Down	16	3	2	2	4	59.26	**28.26**
	Weak Down	12	6	1	5	11	17.14	
	Neutral	14	4	1	3	6	3.57	
	Weak Up	10	2	0	3	8	13	
	Strong Up	2	5	0	5	13	52	
	Precision (%)	29.63	30	25	16.67	30.95		

Table 4.13: Confusion matrix for the LSTM-based model with Conv1D multi-head attention, i.e., best model, on the final test dataset.

Table 4.13 is the confusion matrix for the model predictions on this same final test dataset that produces Table 4.12. The results are congruent; it can be observed that the number of trades instantiated is equal to the number of times a class that results in action is predicted, 134. The total expected green trades in Table 4.13 is 66 and reds is 41, different from the actual result in Table 4.12. This can be explained by the number of times the model predicts some direction, but the real class is Neutral. This will result in a trade instantiating with a small red or green outcome. Also, the classes are feature engineered from the raw value representing the integral of the price evolution during the predicting time. This area can mean a consistent movement to one side, but some rapid, aggressive move to the opposite side can occur. This is another explanation of why the confusion matrix's expected results can differ from what happens in reality. As indicator of the model resilience, is verified that the overall ratio of results remains similar in the production phase (Table 4.13) and also on the validation dataset confusion matrices (Table 4.11).

Summary

In this Chapter, we present in detail new add-ons to formalize new DL architectures, namely roll padding, multi-head attention using Conv1D for LSTM-based models, and multi-head attention using Conv2D with roll padding for ConvLSTM2D. We demonstrate that these add-ons can improve the learning process. In particular, the type of attention conceived for the ConvLSTM2D is a powerful mechanism for cyclical MTS problems. In the betting exchange case study, an end-to-end framework is shown at work. In what follows, we discuss the final conclusions and future research avenues.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

This dissertation proposes applying a new padding method designated by roll padding and introduces new types of attention mechanisms in LSTMs and ConvLSTM2D for MTS problems. Multi-head attention mechanisms split the MTS into individual time series, and create a path of attention for each sequence. Within each sequence path, we introduce convolutional layers to build the attention vector. While Conv1D layers are used in LSTM-based models, Conv2D layers can be used in the ConvLSTM2D model. We also take advantage of roll padding by introducing it into the ConvLSTM2D proposed attention mechanism to capture cyclical properties of the MTS when deemed necessary.

We also extend the WaveNet to work with Conv2D layers rather than with Conv1D layers. By doing so, we can personalize the level of correlation between sequences, to be found by the convolution kernel. While the standard WaveNet combines all sequences of the MTS treating them as channels of feature maps to be processed by the kernel, the proposed extension allows specifying, by the kernel size, how many sequences the kernel should focus on. As such, the standard WaveNet becomes a particular case of the proposed extension. Using Conv2D layers instead of Conv1D in the WaveNet also allows combining roll padding with causal padding, which is the main ingredient of the WaveNet. Overall, all these new particularities improve the performance of DL models in MTS predictions. Another main conclusion of this dissertation is that the chosen architecture is very problem dependent given the heterogeneity of the results.

Human Activity Recognition

The application of a WaveNet 2D with roll padding overcomes all other models in terms of accuracy. The combination of roll padding with causal padding in the WaveNet allows to achieve the best results in the RTD dataset of this case study. Also, accuracy improves with the introduction of roll padding in the basic CNN which serves as a sandbox to demonstrate the potential of roll padding.

Air Pollution - $PM_{2.5}$ Concentration

Despite the roll padding improving base architectures, the best model is the ConvLSTM2D without add-ons, which neither improve nor affect the learning curves. The proposed attention mechanism for the ConvLSTM2D aims at taking advantage of possible cyclic input sequences. Consequently, a potential justification is the inadequate definition of the input sequence size. For this dissertation 72 time steps were defined, where each corresponds to one hour. To capture week cyclic behavior, future work should consider sequence sizes of 168 time steps and similar transformations to what was developed in the household electric power consumption case study.

Household Electric Power Consumption

This case study reveals the positive impact of proposed Conv2D attention mechanisms on the ConvLSTM2D-based model with roll padding. Although the accuracy result may seem relatively low, the main observation that should be considered to make conclusions about the convergence of DL models in MTS classification is approximating the confusion matrix's values to the principal diagonal. To confirm the good performance, we compare it with the classical ARIMAX in regression mode, surpassing it.

Exchange Markets

A complete trading framework was developed for this case study. The pre-live horse race markets were analyzed, which is a challenging environment for prediction. We conclude that the DL model with the best performance is the LSTM with multi-head Conv1D attention. CNN-based models exhibit a low performance, which can be explained by the heavy use of the ReLU activation function that tends to produce sparse weights (see Section 2.1.1). Differently, LSTM-based models avoid the ReLU activation function in the main pipeline, resulting in a thinner learning evolution that is adequate for the hard MTS problem in hand. This issue can also be tackled by analyzing the learning rate in future research. Finally,

we also tested the Cohen's kappa error function normally used for ordered classification problems, but results did not improve.

5.2 Future Work

One of the dissertation's future paths is to direct the work of time series towards sequences of images. The analysis and/or forecasting of image sequence problems has application in the most varied areas, particularly in industrial engineering, where the detection of anomalies (e.g., paper industry) is desirable. In this way, the focus will be on problems of time series of images where there is a continuity of information that should be processed to find anomalies. The objective is to create a DL conceptual model using architectures based on ConvLSTM2D layers for this purpose. This future development aims to model time series of images using, for this purpose, DL methodologies to automatically extract knowledge from a given context and help identify anomalous properties. The final goal is to have a sequence-to-sequence model to generate the expected image. Like AEs, we train this model in a unsupervised manner. This way, it is not needed prior information about the type of anomalies.

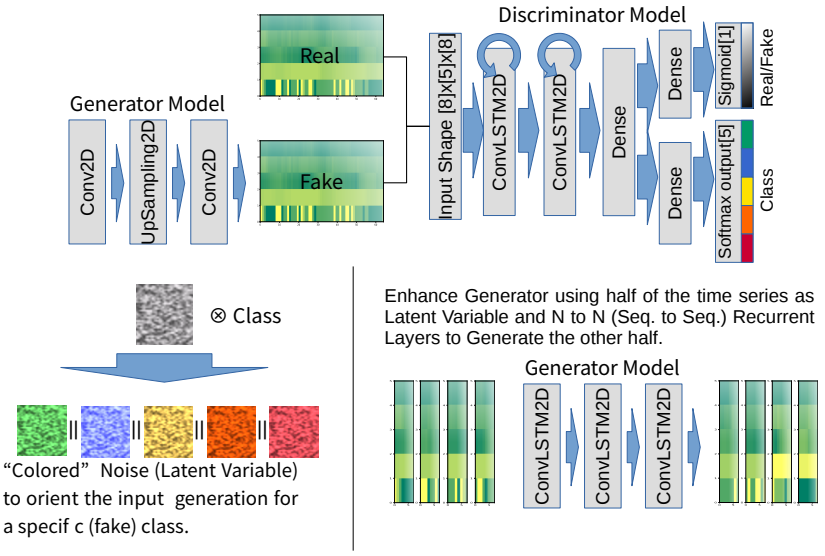


Figure 5.1: Projected Auxiliary Classification Generative Adversarial Network (AC-GAN) architecture for MTS problems.

Another relevant future path is to introduce AC-GANs for synthetic construction of indicators based on context, meaning that experimental methodologies will be developed and tested based on deep generative architectures. Within this research domain, AC-GANs based on sequence-to-sequence generators can provide useful insights into MTS's evolution problems. As highlighted in Figure 5.1, both generator and discriminator use layers that

process sequential information. The generator considers an initial partition of the MTS sequence as the latent space to generate the complementary partition of the MTS sequence, thus, defining a sequence-to-sequence generator. Then, the discriminator decides whether the input information is real or not, and the class where it falls. Ultimately, this methodology will allow the construction of indicators from a given future scenario.

Appendix

Betting Exchange Statistics

	Mean	Std Dev	Min	Max
Volume				
1st min	520865	275541	14242	4858654
2nd min	423930	247598	10783	4713870
3rd min	347772	223834	8390	4569493
4th min	285066	201834	7014	4237354
5th min	236840	183575	5818	3944767
6th min	200076	168577	4514	3779854
7th min	172516	156206	2524	3554934
8th min	150905	145219	1898	3481841
9th min	134880	137576	1647	3450423
10th min	122693	131643	1503	3420282
Liquidity				
1st min	6974	12301	531	451241
2nd min	5967	13151	682	468516
3rd min	4975	12832	526	454538
4th min	4205	12544	393	453012
5th min	3652	12143	329	428307
6th min	3105	11137	285	399753
7th min	2774	10808	245	359948
8th min	2451	9825	212	331043
9th min	2224	9540	191	323377
10th min	2029	8935	171	292921
Volatility^{+}				
1st min	1052	1443	77	29899
2nd min	1100	1593	0	31323
3rd min	1107	1357	22	37893
4th min	1067	1524	0	29442
5th min	984	1412	20	33069
6th min	868	1437	29	49078
7th min	796	1191	4	30815
8th min	729	1162	7	27630
9th min	650	961	2	29686
10th min	642	1326	1	38439

Table A1: Pre-live betfair horse racing markets summary statistics.

Note: Total number of observations (i.e. races): 14421. Each line represents the x minute before the start of a race, $x = \{1st, ..., 10th\}$. Units of measure are clarified in Fig.3.8 .

$^{+}$ Ticks variation in absolute value per minute.

P&L	TM	END_STATE	EVENT	RUNNER	VOLUME	N_R	ENTR	TARG	STO	DIR	T_P	T_L	PT_P	PT_L	O_ODD	C_ODD	O_AM	C_AM
£0,00	2	NOT_OPEN	Ham_2nd_Sep	Nightster	38189.27	8	5.5	6.2	5.1	LB	6	4	£11,29	-£7,84	0	0	£0,00	£0,00
£0,00	2	NOT_OPEN	Ham_2nd_Sep	Tectonic	18415.76	8	4.5	3.95	4.9	BL	6	4	£13,92	-£8,16	0	0	£0,00	£0,00
£6,52	1	CLOSED	FfosL_2nd_Sep	Mccool_Bannanas	23465.96	6	4.9	4.6	5.2	BL	3	3	£6,52	-£5,77	4.9	4.6	£100,00	£106,52
-£2,00	1	CLOSED	Ham_2nd_Sep	King_Of_Paradise	15501.31	6	5.1	5.5	4.7	LB	4	4	£7,27	-£8,51	5.1	5	£100,00	£102,00
£0,00	1	NOT_OPEN	FfosL_2nd_Sep	Cabuchon	22716.31	7	4.4	4.1	4.7	BL	3	3	£7,32	-£6,38	0	0	£0,00	£0,00
£0,00	1	CLOSED	FfosL_2nd_Sep	Men_Dont_Cry	24814.14	7	4.5	4.9	4.1	LB	4	4	£8,16	-£9,76	4.5	4.5	£100,00	£100,00
£0,00	2	NOT_OPEN	Ham_2nd_Sep	George_Fenton	20976.69	9	4.9	4.3	5.3	BL	6	4	£13,95	-£7,55	0	0	£0,00	£0,00
£6,00	1	CLOSED	Brig_2nd_Sep	Admiralofthesea	20249.86	9	5.3	5	5.6	BL	3	3	£6,00	-£5,36	5.3	5	£100,00	£106,00
£0,00	1	CLOSED	Muss_3rd_Sep	Mishaal	21280.96	9	5	4.7	5.3	BL	3	3	£6,38	-£5,66	5	5	£100,00	£100,00
£11,11	2	CLOSED	Good_3rd_Sep	Deeds_Not_Words	21824.36	6	6	5.4	6.8	BL	6	4	£11,11	-£11,76	6	5.4	£100,00	£111,11
£0,00	1	CLOSED	Good_3rd_Sep	Argent_Knight	45269.49	9	5.6	5.3	5.9	BL	3	3	£5,66	-£5,08	5.6	5.6	£100,00	£100,00
-£5,77	2	CLOSED	Good_3rd_Sep	Minority_Interest	15054.53	10	4.9	4.3	5.3	BL	6	4	£13,95	-£7,55	4.9	5.2	£100,00	£94,23
£5,66	2	CLOSED	Good_3rd_Sep	Swift_Blade	19614.89	10	5.6	5	6	BL	6	4	£12,00	-£6,67	5.6	5.3	£100,00	£105,66
£0,00	2	NOT_OPEN	Ling_3rd_Sep	Prospera	30635.19	7	4.8	5.4	4.4	LB	6	4	£11,11	-£9,09	0	0	£0,00	£0,00
£10,71	2	CLOSED	Ling_3rd_Sep	Rock_God	29429.46	7	5	5.6	4.6	LB	6	4	£10,71	-£8,70	5	5.6	£100,00	£89,29
£2,27	2	CLOSED	Bath_4th_Sep	Kakapuka	31543.94	7	4.5	3.95	4.9	BL	6	4	£13,92	-£8,16	4.5	4.4	£100,00	£102,27
£2,04	2	CLOSED	Bath_4th_Sep	Dreams_Of_Glory	31036.05	7	4.8	5.4	4.4	LB	6	4	£11,11	-£9,09	4.8	4.9	£100,00	£97,96
£5,20	2	CLOSED	Bath_4th_Sep	Devon_Diva	6343.06	6	4.3	3.85	4.7	BL	6	4	£11,69	-£8,51	4.3	4	£69,27	£74,47
£0,00	2	CLOSED	Kemp_4th_Sep	For_Posterity	17928.36	8	4.6	4	5	BL	6	4	£15,00	-£8,00	4.6	4.6	£100,00	£100,00
£0,00	2	NOT_OPEN	Salis_5th_Sep	Mysterious_Man	29961.6	6	4.2	3.8	4.6	BL	6	4	£10,53	-£8,70	0	0	£0,00	£0,00
£0,00	2	NOT_OPEN	Salis_5th_Sep	New_Rich	14194.09	6	4.2	3.8	4.6	BL	6	4	£10,53	-£8,70	0	0	£0,00	£0,00
£0,00	1	NOT_OPEN	Salis_5th_Sep	Catchanova	19254.96	7	4.2	3.95	4.5	BL	3	3	£6,33	-£6,67	0	0	£0,00	£0,00
£3,74	1	CLOSED	Salis_5th_Sep	South_Cape	17128.67	7	4.3	4.7	3.95	LB	4	4	£8,51	-£8,86	4.3	4.47	£100,00	£96,26
-£0,00	1	CLOSED	Newc_6th_Sep	Red_Pike	17883.92	10	4.3	4.7	3.95	LB	4	4	£8,51	-£8,86	4.3	4.3	£100,00	£100,00
£0,00	2	CLOSED	Newc_6th_Sep	Noble_Asset	33583.39	9	4.3	4.9	3.95	LB	6	4	£12,24	-£8,86	4.3	4.3	£100,00	£100,00
-£2,13	2	CLOSED	Newc_6th_Sep	Pure_Impressions	11972.8	9	4.6	4	5	BL	6	4	£15,00	-£8,00	4.6	4.7	£100,00	£97,87
£0,00	1	CLOSED	Hayd_6th_Sep	Ashpan_Sam	20740.66	8	4.3	4.7	3.95	LB	4	4	£8,51	-£8,86	4.3	4.3	£93,25	£93,25
£0,00	1	NOT_OPEN	Hayd_6th_Sep	Anomaly	19408.38	8	5.8	5.5	6.2	BL	3	3	£5,45	-£6,45	0	0	£0,00	£0,00
£0,00	2	NOT_OPEN	Kemp_7th_Sep	Masterstroke	26429.99	10	6	5.4	6.8	BL	6	4	£11,11	-£11,76	0	0	£0,00	£0,00
-£3,77	2	CLOSED	Ascot_7th_Sep	Steventon_Star	10016.69	7	5.1	4.5	5.5	BL	6	4	£13,33	-£7,27	5.1	5.3	£100,00	£96,23
£3,64	2	CLOSED	Ascot_7th_Sep	Forgive	19005.23	8	5.3	5.9	4.9	LB	6	4	£10,17	-£8,16	5.3	5.5	£100,00	£96,36
£0,00	1	NOT_OPEN	Wolv_7th_Sep	Lord_Buffhead	48688.37	10	5.8	6.4	5.4	LB	4	4	£9,38	-£7,41	0	0	£0,00	£0,00
£2,44	2	CLOSED	Font_8th_Sep	Brough_Academy	13833.73	7	4.2	3.8	4.6	BL	6	4	£10,53	-£8,70	4.2	4.1	£100,00	£102,44
£2,13	2	CLOSED	Font_8th_Sep	Chilworth_Screamer	18831.11	7	4.8	4.2	5.2	BL	6	4	£14,29	-£7,69	4.8	4.7	£100,00	£102,13
-£0,00	2	CLOSED	Font_8th_Sep	The_Tracey_Shuffle	44132.1	7	4.1	4.7	3.85	LB	6	4	£12,77	-£6,49	4.1	4.1	£100,00	£100,00
-£7,02	1	CLOSED	Font_8th_Sep	Ovilia	47113.2	9	5.3	5	5.6	BL	3	3	£6,00	-£5,36	5.3	5.7	£100,00	£92,98
£0,00	2	NOT_OPEN	Hunt_9th_Sep	Brimham_Boy	19783.33	9	6	7.2	5.6	LB	6	4	£16,67	-£7,14	0	0	£0,00	£0,00
£1,96	1	CLOSED	Hunt_9th_Sep	Tiny_Tenor	10597.64	6	5	5.4	4.6	LB	4	4	£7,41	-£8,70	5	5.1	£100,00	£98,04
-£5,13	2	CLOSED	Perth_9th_Sep	Rathmoyle_House	15105.65	6	4.1	4.7	3.85	LB	6	4	£12,77	-£6,49	4.1	3.9	£100,00	£105,13
£4,55	2	CLOSED	Hunt_9th_Sep	Getaway_Car	43158.07	7	4.6	4	5	BL	6	4	£15,00	-£8,00	4.6	4.4	£100,00	£104,55
-£4,44	2	CLOSED	Brig_9th_Sep	Arlecchino	34162.89	6	4.3	3.85	4.7	BL	6	4	£11,69	-£8,51	4.3	4.5	£100,00	£95,56
£7,14	1	CLOSED	Brig_9th_Sep	Kamchatka	8723.94	6	4.5	4.2	4.8	BL	3	3	£7,14	-£6,25	4.5	4.2	£100,00	£107,14
£2,08	2	CLOSED	Leic_10th_Sep	Tatlisu	32783.13	7	4.9	4.3	5.3	BL	6	4	£13,95	-£7,55	4.9	4.8	£100,00	£102,08
£7,84	2	CLOSED	Worc_10th_Sep	Gud_Day	16815.93	6	4.7	5.3	4.3	LB	6	4	£11,32	-£9,30	4.7	5.1	£100,00	£92,16
£0,00	2	NOT_OPEN	Bev_10th_Sep	Hadaj	19770.65	8	4.3	3.85	4.7	BL	6	4	£11,69	-£8,51	0	0	£0,00	£0,00
-£5,00	1	CLOSED	Bev_10th_Sep	Bondi_Beach_Boy	16922.4	8	5.7	5.4	6	BL	3	3	£5,56	-£5,00	5.7	6	£100,00	£95,00
£10,94	2	CLOSED	Bev_10th_Sep	Dubai_Dynamo	20403.49	8	5.7	6.6	5.3	LB	6	4	£13,64	-£7,55	5.7	6.4	£100,00	£89,06
£0,00	2	NOT_OPEN	Bev_10th_Sep	Starlit_Cantata	35090.79	9	5.1	4.5	5.5	BL	6	4	£13,33	-£7,27	0	0	£0,00	£0,00
-£6,67	2	CLOSED	Uttox_11th_Sep	Teak	19655.05	7	5.6	5	6	BL	6	4	£12,00	-£6,67	5.6	6	£100,00	£93,33
...	...	...	...	...	...	...	...	...	...	...	...	..	...	...	..	...	...	...

Table A2: Stretch of the entire output fields of trading mechanisms (TM 2 - Trailing-stop and 1 - Swing) and parametrization according to the DL models prediction, during several races. The base stake used is £100.00.

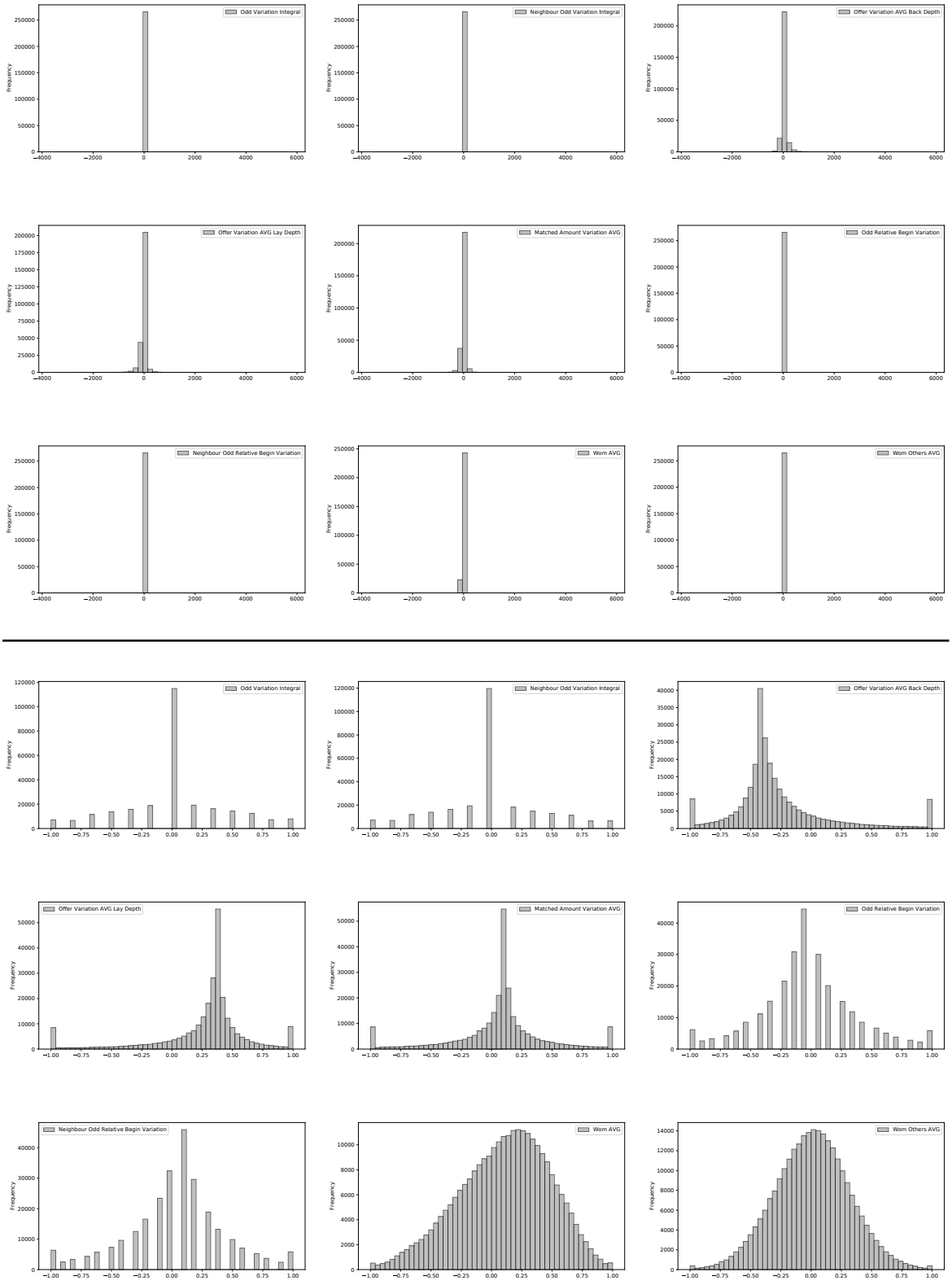


Figure A1: Example of histogram re-scaling with truncated tails at 10% level to find min-max values for input normalization. The illustration represents inputs on category #41 of the rule-based system index of the betting exchange horse race markets case study. Top 9 subplots are raw data and bottom 9 subplots are the result of applying this technique.

Models Main Components

Listing A1: Python Keras layer implmentation of roll pading with causal pading. This layer is used before a convolution with valid/no pading inside WaveNet 2D.

```
class CylindricalPadCausal(Layer):

        def __init__(self, m=0,n=1, **kwargs):
                super(CylindricalPadCausal, self).__init__(**kwargs)
                self.n = n
                self.m = m
                #assert n > 0, 'n must be positive'

        def build(self, input_shape):
                self.input_spec = [InputSpec(shape=input_shape)]
                super(CylindricalPadCausal, self).build(input_shape)

        def compute_output_shape(self, input_shape):
                return (input_shape[0],
                        input_shape[1],
                        # ONLY ADD m 0's to the left for casusal padding
                        input_shape[2] + self.m,
                        input_shape[3] + 2*self.n)

        def get_output_shape_for(self, input_shape):
                return (input_shape[0],
                        input_shape[1],
                        # ONLY ADD m 0's to the left for casusal padding
                        input_shape[2] + self.m,
                        input_shape[3] + 2*self.n)

        def call(self,Element, mask=None):
                firstColumns=Element[:,:,:,0:self.n]
                lastColumns=Element[:,:,:,Element.shape[3]-self.n:Element.shape[3]]
                ## ROLL add end to beginning and beginning to the end
                result=tf.concat([Element,firstColumns], axis=3)
                result=tf.concat([lastColumns,result], axis=3)
                if self.m != 0 :  ### only add 0's to de left for casusal pading
                        firstRows=result[:,:,0:self.m,:]
                        sa=tf.shape(firstRows)[0]
                        sb=tf.shape(firstRows)[1]
                        sc=self.m
                        sd=tf.shape(firstRows)[3]
                        y = tf.fill([sa, sb,sc, sd], 0.)
                        result=tf.concat([y,result], axis=2)
                return result

        def get_config(self):
                config = {'n': self.n,
                'm': self.m}
                base_config = super(CylindricalPadCausal, self).get_config()
                return dict(list(base_config.items()) + list(config.items()))
```

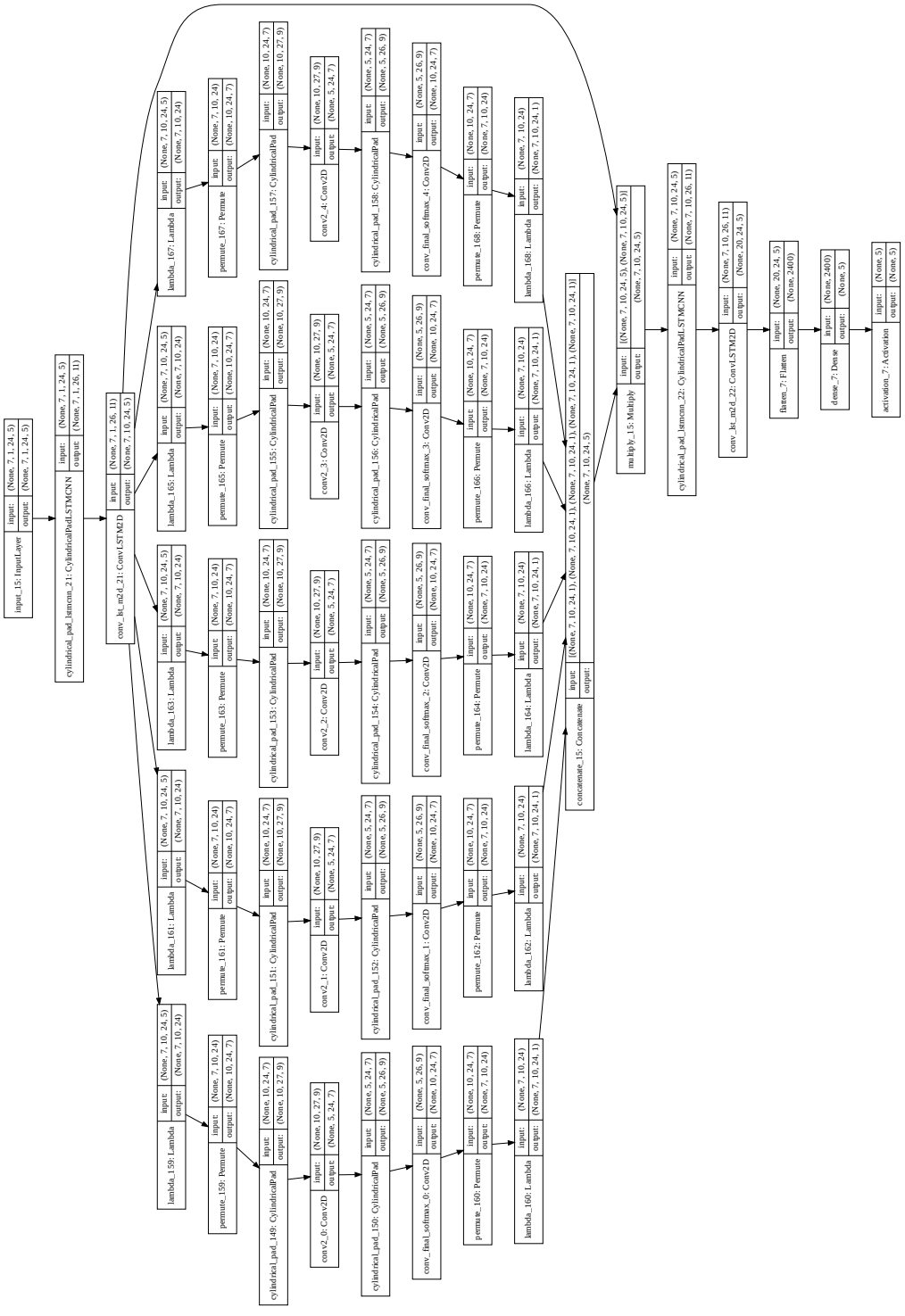


Figure A2: Stacked ConvLSTM2D with roll padding, using input of: 7 segments/days, 1 channel, 24h/steps, and 5 variables. Multi-head attention blocks per variable with two convolution layers using 2D maps of 7 segments/days × 24h/steps. Inside the attention block roll padding is applied over the 7 days dimension. Architecture used for energy house consumption prediction. (The case study has 8 variables; here, we plot the model with 5 variables for simplicity.)

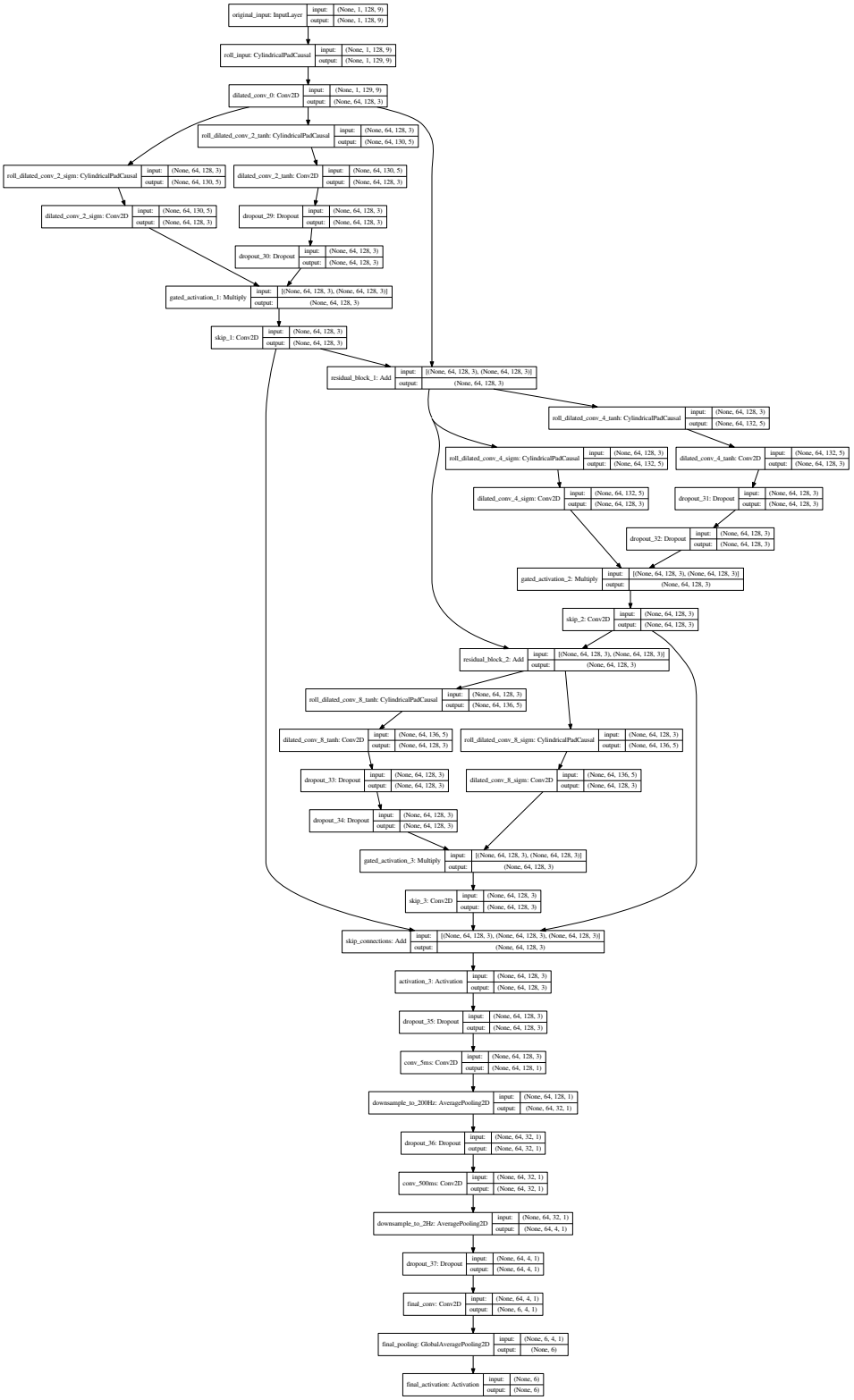


Figure A3: Wavenet 2D with depth 3 using inputs of: 128 steps $\times$ 9 variables. It uses roll padding on the variable component and causal padding in the time steps component. Model used in the HAR case study. (The case study has residual blocks in depth 8; here, we plot the model with 3 blocks for simplicity.)

Glossary

AC-GAN Auxiliary Classification Generative Adversarial Network. xx, 109

AdaGrad Adaptive Gradient algorithm. 40

Adam Adaptive Moment Estimation. 11, 14, 15, 40, 41

AE Autoencoder. 11, 37, 38, 109

AFE Automated Feature Engineering. 11, 37

AI Artificial Intelligence. xvii, 5, 52

ANN Artificial Neural Network. 4, 7, 19, 55

API Application Programming Interface. 62, 66, 67, 73

ARIMA Auto-Regressive Integrated Moving Average. 47, 49, 55

BPTT backpropagation through time. 28, 31

CNN Convolutional Neural Network. xviii, xix, 10, 11, 25, 31, 32, 38, 53–55, 58, 83–86, 94–98, 100, 101, 108

ConvLSTM2D Bi-dimensional Convolutional LSTM. xv, xvi, xix, xx, 10, 34, 90, 91, 96–101, 105, 107–109, 115

DFT Discrete Fourier Transform. 53, 54

DL Deep Learning. 4–7, 9, 12, 15, 17, 34, 38, 49, 51, 52, 55, 58, 59, 75, 79, 81, 83, 92, 95, 98–102, 105, 107–109

DNN Deep Neural Network. 12, 20, 25, 43

FE Feature Engineering. 10, 11, 13, 51–54, 56, 60, 81

FFNN Feed Forward Neural Network. xvii, xviii, 13, 15, 16, 19, 21, 27–31, 38

FFT Fast Fourier Transform. 53, 54

FIFO First In First Out. 64

GA Genetic Algorithm. 41–44

GAN Generative Adversarial Network. 34

HAR Human Activity Recognition. xv, xviii, xx, 10, 51–54, 81, 116

LSTM Long Short-Term Memory. xvi, xviii, xix, 5, 10, 14, 29–31, 34, 36, 38, 55, 58, 86–90, 94–102, 105, 107, 108

LVQ Learning Vector Quantization. 53, 54

ML Machine Learning. 1, 3, 4, 8, 12, 51, 52, 55, 58

MSE Mean Square Error. 12, 37, 46

MTS Multivariate Time Series. xvii–xx, 6, 7, 10, 19, 29, 32, 34, 47, 49, 51–55, 59, 60, 81, 84–86, 88–94, 99, 105, 107–110

NN Neural Network. xviii, 4, 5, 7–9, 11–15, 19–22, 25–29, 31, 34, 36–47, 49, 59, 75, 79, 83, 87, 88, 90

OVA One-vs-All. 9, 54

OVO One-vs-One. 9, 53, 54

PCA Principal Component Analysis. 14, 38

PL Profit & Loss. 62, 63, 65, 69, 70, 80

$PM_{2.5}$ Particulate Matter 2.5. 10, 51, 56, 57, 81

PSO Particle Swarm Optimization. xviii, 44, 45

RDF Raw Data Frame. 61, 74, 76, 77

ReLU Rectified Linear Unit. xviii, 23, 25, 26, 95, 108

RMSE Root Mean Square Error. 12, 46, 52

RMSprop Root Mean Square Propagation. 15

RNN Recurrent Neural Network. 14, 27–31, 35, 36, 86, 87

RTD Raw Time Distributed. 52–54, 108

SAE Stacked Autoencoder. 38

SGD Stochastic Gradient Descent. 15

SRN Simple Recurrent NN. xviii, 15, 27, 28

SVM Support Vector Machine. 9, 53–55

TCN Temporal Convolutional Network. xviii, 36

UCI University of California Irvine. xv, 10, 51–54, 58, 81

VAE Variational Autoencoder. 34

WoM Weight Of Money. 76, 77

WPI Wholesale Price Index. xvii, 8, 9

XOR Exclusive Or. xvii, 19, 20

References

KP Amber, MW Aslam, and SK Hussain. Electricity consumption forecasting models for administration buildings of the uk higher education sector. *Energy Build 2015;*, 90::127–36, 2015.

Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge Luis Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones. In *Esann*, 2013.

Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding deep neural networks with rectified linear units. *CoRR*, abs/1611.01491, 2016. URL `http://arxiv.org/abs/1611.01491`.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling, 2018, 2018.

Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 37–49, 2012.

Alejandro Baldominos, Yago Saez, and Pedro Isasi. On the automated, evolutionary design of neural networks: past, present, and future. *Neural Computing and Applications*, pages 1–27, 2020.

Snehasree Behera, Bhawani Shankar Pattnaik, Motahar Reza, and D. S. Roy. Predicting consumer loads for improved power scheduling in smart homes. In Himansu Sekhar Behera and Durga Prasad Mohapatra, editors, *Computational Intelligence in Data Mining—Volume 2*, pages 463–473, New Delhi, 2016. Springer India. ISBN 978-81-322-2731-1.

Beijing US Embassy. Beijing PM2.5 Data Data Set , December 2014. `https://archive.ics.uci.edu/ml/datasets/Beijing+PM2.5+Data`.

Y. Bengio. *Learning Deep Architectures for AI*. Essence of knowledge, The. Now Publishers, 2009. ISBN 9781601982940. URL `https://books.google.pt/books?id=cq5ewg7FniMC`.

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, 5(2):157–166, March 1994. ISSN 1045-9227. doi: 10.1109/72.279181. URL http://dx.doi.org/10.1109/72.279181.

Yoshua Bengio. Deep learning of representations for unsupervised and transfer learning. In Isabelle Guyon, Gideon Dror, Vincent Lemaire, Graham Taylor, and Daniel Silver, editors, *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, volume 27 of *Proceedings of Machine Learning Research*, pages 17–36, Bellevue, Washington, USA, 02 Jul 2012. PMLR. URL http://proceedings.mlr.press/v27/bengio12a.html.

Betfair. *Sports API Reference Guide - v1.101*. The Sports Exchange API Documentation. Betfair, March 2012.

Léon Bottou. Stochastic gradient descent tricks. In Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade - Second Edition*, volume 7700 of *Lecture Notes in Computer Science*, pages 421–436. Springer, 2012. ISBN 978-3-642-35288-1. doi: 10.1007/978-3-642-35289-8_25. URL https://doi.org/10.1007/978-3-642-35289-8_25.

George Edward Pelham Box and Gwilym Jenkins. *Time Series Analysis, Forecasting and Control*. Holden-Day, Inc., San Francisco, CA, USA, 1990. ISBN 0816211043.

John F. Carter. *Mastering the Trade: Proven Techniques for Profiting from Intraday and Swing Trading Setups*. American Media International, 2007. ISBN 1933309628.

Yiling Chen, Sharad Goel, and David M. Pennock. Pricing combinatorial markets for tournaments. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, STOC '08, pages 305–314, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-047-0. doi: 10.1145/1374376.1374421.

Yuanyi Chen, Yubin Wang, and Qiang Yang. Cascaded denoising convolutional auto-encoders for automatic recovery of missing time series data. In *2020 19th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES)*, pages 283–286. IEEE, 2020.

Bing Cheng and D. M. Titterington. Neural networks: A review from a statistical perspective. *Statist. Sci.*, 9(1):2–30, 02 1994. doi: 10.1214/ss/1177010638. URL https://doi.org/10.1214/ss/1177010638.

Pasapitch Chujai, Nittaya Kerdprasop, and Kittisak Kerdprasop. Time series analysis of household electric consumption with arima and arma models. In *Proceedings of the International MultiConference of Engineers and Computer Scientists*, volume 1, pages 295–300, 2013.

Gali Cohen, Ilan Levy, Yuval, Jeremy D Kark, Noam Levin, David M Broday, David M Steinberg, and Yariv Gerber. Long-term exposure to traffic-related air pollution and cancer among survivors of myocardial infarction: A 20-year follow-up study. *European Journal of Preventive Cardiology*, 24(1):92–102, 2017.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, Dec 1989. ISSN 1435-568X. doi: 10.1007/BF02551274. URL `https://doi.org/10.1007/BF02551274`.

Patrícia de Oliveira e Lucas, Marcos Antonio Alves, Petrônio Cândido de Lima e Silva, and Frederico Gadelha Guimar aes. Reference evapotranspiration time series forecasting with ensemble of convolutional neural networks. *Computers and Electronics in Agriculture*, 177: 105700, 2020. ISSN 0168-1699. doi: https://doi.org/10.1016/j.compag.2020.105700. URL `http://www.sciencedirect.com/science/article/pii/S016816992031262X`.

G. Deboeck. *Trading on the Edge: Neural, Genetic, and Fuzzy Systems for Chaotic Financial Markets*. Wiley Finance. Wiley, 1994. ISBN 9780471311003.

Dwight Deugo, Michael Weiss, and Elizabeth Kendall. Reusable patterns for agent coordination. In *in: Omicini, A., Coordination of Internet Agents*, pages 347–368. Springer, 2001.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=1953048.2021068`.

Timothy S Dye. Guidelines for developing an air quality (ozone and pm2. 5) forecasting program. 2003.

Jeffrey L. Elman. Finding structure in time. *COGNITIVE SCIENCE*, 14(2):179–211, 1990.

Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *J. Mach. Learn. Res.*, 11:625–660, March 2010. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=1756006.1756025`.

Kevin Fauvel, Tao Lin, Véronique Masson, Élisa Fromont, and Alexandre Termier. Xcm: An explainable convolutional neural network for multivariate time series classification, 2020.

Björn Fischer and Christophe Planas. Large scale fitting of regression models with arima errors. *Journal of Official Statistics*, 16(2):173, 2000.

Nelson Fumo and MA Rafe Biswas. Regression analysis for prediction of residential energy consumption. *Renew Sustain Energy Rev 2015;*, 47::332–43, 2015.

Alice Berard Georges Hebrail. Individual household electric power consumption Data Set, November 2010. http://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL http://proceedings.mlr.press/v9/glorot10a.html.

David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, New York, 1989.

David E. Goldberg. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 2002. ISBN 1402070985.

Rui Goncalves, Vitor Miguel Ribeiro, Fernando Lobo Pereira, and Ana Paula Rocha. Deep learning in exchange markets. *Information Economics and Policy*, 47:38 – 51, 2019. ISSN 0167-6245. doi: https://doi.org/10.1016/j.infoecopol.2019.05.002. URL http://www.sciencedirect.com/science/article/pii/S0167624518300702. The Economics of Artificial Intelligence and Machine Learning.

Rui Gonçalves, AnaPaula Rocha, and FernandoLobo Pereira. High level architecture for trading agents in betting exchange markets. In *Advances in Information Systems and Technologies*, volume 206 of *Advances in Intelligent Systems and Computing*, pages 497–510. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-36980-3. doi: 10.1007/978-3-642-36981-0_46. URL http://dx.doi.org/10.1007/978-3-642-36981-0_46.

Rui Gonçalves, Vitor Ribeiro, Fernando Pereira, and Ana Paula. Roll padding and wavenet for multivariate time series in human activity recognition. In *World Conference on Information Systems and Technologies*, WorldCist'21 - 9th conference. Springer Berlin Heidelberg - AISC series, 2021.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014. URL http://arxiv.org/abs/1410.5401.

Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, 2018.

Calvin Janitra Halim and Kazuhiko Kawamoto. 2d convolutional neural markov models for spatiotemporal sequence forecasting. *Sensors*, 20(15):4195, 2020.

L. G. C. Hamey. A functional approach to border handling in image processing. In *2015 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, pages 1–8, Nov 2015. doi: 10.1109/DICTA.2015.7371214.

James D Hamilton, Daniel F Waggoner, and Tao Zha. Normalization in econometrics. *Econometric Rev 2007;*, 26(2-4)::221–52, 2007.

Arthur A Hancock, Eugene N Bush, Dusanka Stanisic, John J Kyncl, and C Thomas Lin. Data normalization before statistical analysis: keeping the horse before the cart. *Trends in pharmacological sciences 1988;*, 9(1)::29–32, 1988.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

Jeff Heaton. Encog: Library of interchangeable machine learning models for java and c#. *Journal of Machine Learning Research*, 16:1243–1247, 2015. URL `http://jmlr.org/papers/v16/heaton15a.html`.

Donald O. Hebb. *The organization of behavior: A neuropsychological theory*. Wiley, New York, June 1949. ISBN 0-8058-4300-0.

G Hebrail and A Berard. Individual household electric power consumption data set: Uci machine learning repository, school of information and computer science; 2012. 2012.

Keith W. Hipel and A. Ian. McLeod. *Time series modelling of water resources and environmental systems / Keith W. Hipel, A. Ian McLeod*. Elsevier Amsterdam ; New York, 1994. ISBN 0444892702.

Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. Master's thesis, Technical University Munich, Institute of Computer Science, 7 1991. (diploma thesis).

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9 (8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL `http://dx.doi.org/10.1162/neco.1997.9.8.1735`.

John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975. second edition, 1992.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Netw.*, 4(2):251–257, March 1991. ISSN 0893-6080. doi: 10.1016/0893-6080(91)90009-T. URL `http://dx.doi.org/10.1016/0893-6080(91)90009-T`.

Xianxu Hou, LinLin Shen, Ke Sun, and Guoping Qiu. Deep feature consistent variational autoencoder. *CoRR*, abs/1610.00291, 2016. URL `http://arxiv.org/abs/1610.00291`.

Chia-Yu Hsu, Wei-Chen Liu, et al. Multiple time-series convolutional neural network for fault detection and diagnosis and empirical study in semiconductor manufacturing. *Journal of Intelligent Manufacturing*, pages 1–14, 2020.

Andrey Ignatov. Real-time human activity recognition from accelerometer data using convolutional neural networks. *Applied Soft Computing*, 62:915–922, 2018.

Jehn-Ruey Jiang, Juei-En Lee, and Yi-Ming Zeng. Time series multiple channel convolutional neural network with attention-based long short-term memory for predicting bearing remaining useful life. *Sensors*, 20(1):166, 2020.

Wenchao Jiang and Zhaozheng Yin. Human activity recognition using wearable sensors by deep convolutional neural networks. In *Proceedings of the 23rd ACM International Conference on Multimedia*, MM '15, pages 1307–1310, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3459-4. doi: 10.1145/2733373.2806333. URL `http://doi.acm.org/10.1145/2733373.2806333`.

Michael I. Jordan. Serial order: A parallel, distributed processing approach. In Jeffrey L. Elman and David E. Rumelhart, editors, *Advances in Connectionist Theory: Speech*. Erlbaum, Hillsdale, NJ, 1990.

Marika Kaden, Marc Strickert, and Thomas Villmann. A sparse kernelized matrix learning vector quantization model for human activity recognition. In *ESANN*, 2013.

Kostas Karatzas, Nikos Katsifarakis, Cezary Orlowski, and Arkadiusz Sarzyński. Urban air quality forecasting: A regression and a classification approach. In Ngoc Thanh Nguyen, Satoshi Tojo, Le Minh Nguyen, and Bogdan Trawiński, editors, *Intelligent Information and Database Systems*, pages 539–548, Cham, 2017. Springer International Publishing. ISBN 978-3-319-54430-4.

Harveen Kaur and Sachin Ahuja. Time series analysis and prediction of electricity consumption of health care institution using arima model. In Kusum Deep, Jagdish Chand Bansal, Kedar Nath Das, Arvind Kumar Lal, Harish Garg, Atulya K. Nagar, and Millie Pant, editors, *Proceedings of Sixth International Conference on Soft Computing for Problem Solving*, pages 347–358, Singapore, 2017. Springer Singapore. ISBN 978-981-10-3325-4.

P. Kefalas, Holcombe M., Eleftherakis G., and Gheorghe M. Formal development of reactive agent-based systems. In *Encyclopedia of Information Science and Technology, Second Edition, IGI Global*. IAV2007, 2009.

James Kennedy and Russell C. Eberhart. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, Perth, Australia, IEEE Service Center, Piscataway, NJ, 1995.

Zulfiqar Ahmad Khan, Tanveer Hussain, Amin Ullah, Seungmin Rho, Miyoung Lee, and Sung Wook Baik. Towards efficient electricity forecasting in residential and commercial buildings: A novel hybrid cnn with a lstm-ae based framework. *Sensors*, 20(5):1399, 2020.

Tae-Young Kim and Sung-Bae Cho. Predicting the household power consumption using cnn-lstm hybrid networks. In Hujun Yin, David Camacho, Paulo Novais, and Antonio J. Tallón-Ballesteros, editors, *Intelligent Data Engineering and Automated Learning – IDEAL 2018*, pages 481–490, Cham, 2018. Springer International Publishing. ISBN 978-3-030-03493-1.

Tae-Young Kim and Sung-Bae Cho. Predicting residential energy consumption using cnn-lstm neural networks. *Energy 2019;*, 182::72–81, 2019.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

Gebhard Kirchgässner and Jürgen Wolters. *Introduction to Modern Time Series Analysis*. Springer, Berlin, 2007. URL `https://www.alexandria.unisg.ch/52116/`.

D Koschwitz, J Frisch, and C Van Treeck. Data-driven heating and cooling load predictions for non-residential buildings based on support vector machine regression and narx recurrent neural network: A comparative study on district scale. *Energy 2018;*, 165:: 134–42, 2018.

S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1): 79–86, 1951.

Ujjwal Kumar and VK Jain. Arima forecasting of ambient air pollutants (o 3, no, no 2 and co). *Stochastic Environmental Research and Risk Assessment*, 24(5):751–760, 2010.

Y. LeCun and Y. Bengio. Word-level training of a handwritten word recognizer based on convolutional neural networks. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 3 - Conference C: Signal Processing (Cat. No.94CH3440-5)*, volume 2, pages 88–92 vol.2, Oct 1994. doi: 10.1109/ICPR.1994.576881.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. ISSN 0018-9219. doi: 10.1109/5.726791.

Y. LeCun, K. Kavukcuoglu, and C. Farabet. Convolutional networks and applications in vision. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 253–256, May 2010. doi: 10.1109/ISCAS.2010.5537907.

Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2323, 1998a. ISSN 0018-9219. doi: 10.1109/5.726791.

Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, pages 9–50, London, UK, UK, 1998b. Springer-Verlag. ISBN 3-540-65311-2. URL `http://dl.acm.org/citation.cfm?id=645754.668382`.

Yann LeCun, Fu Jie Huang, and Léon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR'04, pages 97–104, Washington, DC, USA, 2004. IEEE Computer Society. URL `http://dl.acm.org/citation.cfm?id=1896300.1896315`.

Xiang Li, Ling Peng, Xiaojing Yao, Shaolong Cui, Yuan Hu, Chengzeng You, and Tianhe Chi. Long short-term memory neural network for air pollutant concentration predictions: Method development and evaluation. *Environmental Pollution*, 231:997 – 1004, 2017. ISSN 0269-7491. doi: https://doi.org/10.1016/j.envpol.2017.08.114. URL `http://www.sciencedirect.com/science/article/pii/S0269749117307534`.

Xuan Liang, Tao Zou, Bin Guo, Shuo Li, Haozhe Zhang, Shuyi Zhang, Hui Huang, and Song Xi Chen. Assessing beijing's pm2. 5 pollution: severity, weather impact, apec and winter heating. *Proc. R. Soc. A*, 471(2182):20150257, 2015.

Cheng-Yuan Liou, Wei-Chen Cheng, Jiun-Wei Liou, and Daw-Ran Liou. Autoencoder for words. *Neurocomput.*, 139:84–96, September 2014. ISSN 0925-2312. doi: 10.1016/j.neucom.2013.09.055. URL `http://dx.doi.org/10.1016/j.neucom.2013.09.055`.

Jeffrey W Lockhart and Gary M Weiss. Limitations with activity recognition methodology & data sets. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, pages 747–756. ACM, 2014.

J. Magee, N. Dulay, and J. Kramer. A constructive development environment for parallel and distributed programs. In *Proceedings of 2nd International Workshop on Configurable Distributed Systems*, pages 4–14, Mar 1994. doi: 10.1109/IWCDS.1994.289940.

Warren Mcculloch and Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147, 1943.

Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969.

Mohammad Amin Morid, Olivia R. Liu Sheng, Kensaku Kawamoto, and Samir Abdelrahman. Learning hidden patterns from patient multivariate time series data using convolutional neural networks: A case study of healthcare cost prediction. *Journal of Biomedical Informatics*, 111:103565, 2020. ISSN 1532-0464. doi: https://doi.org/10.1016/j.jbi.2020.103565. URL `http://www.sciencedirect.com/science/article/pii/S1532046420301933`.

Michael C. Mozer. Backpropagation. In Yves Chauvin and David E. Rumelhart, editors, *Backpropagation*, chapter A Focused Backpropagation Algorithm for Temporal Pattern Recognition, pages 137–169. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1995. ISBN 0-8058-1259-8. URL `http://dl.acm.org/citation.cfm?id=201784.201791`.

Yurii Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2). In *Soviet Mathematics Doklady*, volume 27, pages 372–376, 1983.

Andrew Y. Ng. What data scientists should know about deep learning. In *Extract Data Conference*, 2015.

D. H. Nguyen and B. Widrow. Neural networks for self-learning control systems. *IEEE Control Systems Magazine*, 10(3):18–23, April 1990. ISSN 0272-1708. doi: 10.1109/37.55119.

Esteban Pardo and Norberto Malpica. Air quality forecasting in madrid using long short-term memory networks. In José Manuel Ferrández Vicente, José Ramón Álvarez-Sánchez, Félix de la Paz López, Javier Toledo Moreo, and Hojjat Adeli, editors, *Biomedical Applications Based on Natural and Artificial Computing*, pages 232–239, Cham, 2017. Springer International Publishing. ISBN 978-3-319-59773-7.

Ali Osman Pektas and H. Kerem Cigizoglu. Ann hybrid model versus arima and arimax models of runoff coefficient. *Journal of Hydrology*, 500:21 – 36, 2013. ISSN 0022-1694. doi: https://doi.org/10.1016/j.jhydrol.2013.07.020. URL `http://www.sciencedirect.com/science/article/pii/S0022169413005374`.

Leyland F Pitt, Richard T Watson, and Daniel M Shapiro. Www. betfair. com: World wide wagering. *Communications of the Association for Information Systems (Volume 15)*, 15: 149–161, 2005.

Riccardo Poli. Analysis of the publications on the applications of particle swarm optimisation. *J. Artif. Evol. App.*, 2008:4:1–4:10, January 2008. ISSN 1687-6229. doi: 10.1155/2008/685175. URL `http://dx.doi.org/10.1155/2008/685175`.

Jordan B Pollack. Recursive distributed representations. *Artificial Intelligence*, 46(1-2): 77–105, 1990.

B.T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1 – 17, 1964. ISSN 0041-5553. doi: https://doi.org/10.1016/0041-5553(64)90137-5. URL http://www.sciencedirect.com/science/article/pii/0041555364901375.

Zhen Qin, Yibo Zhang, Shuyu Meng, Zhiguang Qin, and Kim-Kwang Raymond Choo. Imaging and fusing time series for wearable sensor-based human activity recognition. *Information Fusion*, 53:80 – 87, 2020. ISSN 1566-2535. doi: https://doi.org/10.1016/j.inffus.2019.06.014. URL http://www.sciencedirect.com/science/article/pii/S1566253519302180.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015. URL http://arxiv.org/abs/1511.06434.

Martin Riedmiller and Heinrich Braun. Rprop - a fast adaptive learning algorithm. Technical report, Proc. of ISCIS VII), Universitat, 1992.

A. J. Robinson and Frank Fallside. The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Engineering Department, Cambridge University, Cambridge, UK, 1987.

Bernardino Romera-Paredes, Min SH Aung, and Nadia Bianchi-Berthouze. A one-vs-one classifier ensemble with majority voting for activity recognition. In *ESANN*, 2013.

Charissa Ann Ronao and Sung-Bae Cho. Human activity recognition with smartphone sensors using deep learning neural networks. *Expert Syst. Appl.*, 59:235–244, 2016.

F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, Washington, 1962.

D.E. Rumelhart, G.E. Hinton, R.J. Williams, and San Diego. Institute for Cognitive Science University of California. *Learning Internal Representations by Error Propagation*. ICS report. Institute for Cognitive Science, University of California, San Diego, 1985. URL https://books.google.pt/books?id=Ff9iHAAACAAJ.

Ana Russo, Pedro G. Lind, Frank Raischel, Ricardo Trigo, and Manuel Mendes. Neural network forecast of daily pollution concentration using optimal meteorological data at synoptic and local scales. *Atmospheric Pollution Research*, 6(3):540 – 549, 2015. ISSN 1309-1042. doi: https://doi.org/10.5094/APR.2015.060. URL http://www.sciencedirect.com/science/article/pii/S1309104215302245.

Samir K Safi and Ehab A Abu Saif. Using gls to generate forecasts in regression models with auto-correlated disturbances with simulation and palestinian market index data. *American Journal of Theoretical and Applied Statistics*, 3(1):6–17, 2014.

Hasim Sak, Andrew W. Senior, and Françoise Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *CoRR*, abs/1402.1128, 2014. URL `http://arxiv.org/abs/1402.1128`.

Akash Saxena and Shalini Shekhawat. Ambient air quality classification by grey wolf optimizer based support vector machine. *Journal of environmental and public health*, 2017, 2017.

Robert P. Schumaker, Osama K. Solieman, and Hsinchun Chen. Sports data mining. In *Integrated Series in Information Systems 26*. Springer, 2010. ISBN 978-1-4419-6729-9.

Lianyong Qi Shaohua Wan, Xiaolong Xu, Chao Tong, and Zonghua Gu. Deep learning models for real-time human activity recognition with smartphones. *Mobile Netw Appl 25*, page 743 – 755, 2020. doi: 10.14569/IJACSA.2019.0100311. URL `https://doi.org/10.1007/s11036-019-01445-x`.

A. Sharma, Y. Lee, and W. Chung. High accuracy human activity monitoring using neural network. In *2008 Third International Conference on Convergence and Hybrid Information Technology*, volume 1, pages 430–435, Nov 2008. doi: 10.1109/ICCIT.2008.394.

Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai kin Wong, and Wang chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting, 2015.

Fernando M. Silva and Luís B. Almeida. Acceleration techniques for the backpropagation algorithm. In Luis B. Almeida and Christian J. Wellekens, editors, *Neural Networks*, pages 110–119, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg. ISBN 978-3-540-46939-1.

Salwa O. Slim, Ayman Atia, Marwa M.A. Elfattah, and Mostafa-Sami M.Mostafa. Survey on human activity recognition based on acceleration data. *International Journal of Advanced Computer Science and Applications*, 10(3), 2019. doi: 10.14569/IJACSA.2019.0100311. URL `http://dx.doi.org/10.14569/IJACSA.2019.0100311`.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014. URL `http://arxiv.org/abs/1412.6806`.

S. Sridhar and A. Kalaivani. A survey on methodologies for handling imbalance problem in multiclass classification. In P. Suresh, U. Saravanakumar, and Mohammed Saleh Hussein

Al Salameh, editors, *Advances in Smart System Technologies*, pages 775–790, Singapore, 2020. Springer Singapore. ISBN 978-981-15-5029-4.

Qiang Sun, Yanmin Zhu, Xiaomin Chen, Ailan Xu, and Xiaoyan Peng. A hybrid deep learning model with multi-source data for pm 2.5 concentration forecast. *Air Quality, Atmosphere & Health*, pages 1–11, 2020.

Ola M Surakhi, Martha Arbayani Zaidan, Sami Serhan, Imad Salah, and Tareq Hussein. An optimal stacked ensemble deep learning model for predicting time-series data using a genetic algorithm—an application for aerosol particle number concentrations. *Computers*, 9(4):89, 2020.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

H. Tong. *Threshold models in non-linear time series analysis*. Lecture notes in statistics. Springer-Verlag, 1983. ISBN 9780387909189. URL `https://books.google.pt/books?id=_hTvAAAAMAAJ`.

Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2(42):230–265, 1936. URL `http://www.cs.helsinki.fi/u/gionis/cc05/OnComputableNumbers.pdf`.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

Chi-Man Vong, Weng-Fai Ip, Pak-kin Wong, and Jing-yi Yang. Short-term prediction of air pollution in macau using support vector machines. *Journal of Control Science and Engineering*, 2012, 2012.

Jin-Feng Wang, Mao-Gui Hu, Cheng-Dong Xu, George Christakos, and Yu Zhao. Estimation of citywide air pollution in beijing. *PLOS ONE*, 8(1):1–6, 01 2013. doi: 10.1371/journal.pone.0053400. URL `https://doi.org/10.1371/journal.pone.0053400`.

Yanwen Wang, Yiqun Han, Tong Zhu, Weiju Li, and Hongyin Zhang. A prospective study (scope) comparing the cardiometabolic and respiratory effects of air pollution exposure on healthy and pre-diabetic individuals. *Science China Life Sciences*, 61(1):46–56, 2018.

Paul J. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1(4):339 – 356, 1988. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(88)90007-X. URL http://www.sciencedirect.com/science/article/pii/089360808890007X.

P.J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Harvard University, 1975. URL https://books.google.pt/books?id=z81XmgEACAAJ.

JM Wooldridge. Introductory econometrics: A modern approach, 2016. Nelson Education., 2016.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

Stephen S Yau, Doo-Hwan Bae, and Jun Wang. An architecture-independent software development approach for parallel processing systems. In *Computer Software and Applications Conference, 1995. COMPSAC 95. Proceedings., Nineteenth Annual International*, pages 370–375. IEEE, 1995.

Chunyong Yin, Sun Zhang, Jin Wang, and Neal N Xiong. Anomaly detection based on convolutional recurrent autoencoder for iot time series. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2020.

Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012. URL http://arxiv.org/abs/1212.5701.

Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 818–833, Cham, 2014. Springer International Publishing. ISBN 978-3-319-10590-1.

An Zhang, Qingwen Qi, Lili Jiang, Fang Zhou, and Jinfeng Wang. Population exposure to pm2.5 in the urban area of beijing. *PLOS ONE*, 8(5):1–9, 05 2013. doi: 10.1371/journal.pone.0063486. URL https://doi.org/10.1371/journal.pone.0063486.

G. Peter Zhang. A neural network ensemble method with jittered training data for time series forecasting. *Inf. Sci.*, 177(23):5329–5346, December 2007. ISSN 0020-0255. doi: 10.1016/j.ins.2007.06.015. URL http://dx.doi.org/10.1016/j.ins.2007.06.015.

Guoqiang Peter Zhang. Time series forecasting using a hybrid arima and neural network model. *Neurocomputing*, 50:159–175, 2003. URL `http://dblp.uni-trier.de/db/journals/ijon/ijon50.html#Zhang03`.

Yang Zhang, Marc Bocquet, Vivien Mallet, Christian Seigneur, and Alexander Baklanov. Real-time air quality forecasting, part i: History, techniques, and current status. *Atmospheric Environment*, 60:632 – 655, 2012. ISSN 1352-2310. doi: https://doi.org/10.1016/j.atmosenv.2012.06.031. URL `http://www.sciencedirect.com/science/article/pii/S1352231012005900`.

O. Zweigle, U. P. Kappeler, K. Haussermann, and P. Levi. Event based distributed real-time communication architecture for multi-agent systems. In *5th International Conference on Computer Sciences and Convergence Information Technology*, pages 503–510, Nov 2010. doi: 10.1109/ICCIT.2010.5711108.