UNIVERSITÀ DEGLI STUDI DI NAPOLI
"FEDERICO II"

Scuola Politecnica e delle Scienze di Base

Area Didattica di Scienze Matematiche Fisiche e Naturali

Dipartimento di Fisica "Ettore Pancini"

Laurea Magistrale in Data Science

NEURAL SEQUENCE ANALYSIS TOOLBOX

Relatori:
Prof. Giuseppe Longo
Dr. Tobias Hecking -
German Aerospace Center (DLR)

Candidato:
Antonio Elia Pascarella
Matr. *P*37000003

Anno Accademico 2020/2021

Abstract

Time series have always been of great interest in the financial sector but today with the advent of sensors and the IoT they have received new attention and their analysis is no longer carried out using linear methods of classical statistics but deep learning is revealing a new paradigm with interesting performances for tasks such as predicting time sequences over time or looking for anomalous patterns that could represent failure of the industrial apparatus. Strategies for time series preprocessing with splines and wavelets are investigated with the present work. Methods such as error based methods and GANs for anomaly detection are also studied and models such as sequence to sequence learning and attention mechanisms for forecasting are taken into consideration. Experiments have been carried out to compare all these methodologies using public data from NASA and airpollution dataset (you can find the links in the experiments chapter). Regarding anomaly detection, the most promising approach was that of GANs. The problem of finding a number of timestamps on which to obtain reliable predictions was also investigated and the problem was formulated in such a way that the neural network itself in the training process can learn the length of the time horizon on which to make predictions. A toolbox has been produced that allows the user to preprocess multivariate time series and implement outlier detection or forecasting applications with the above methodologies.

Dedication

A mia madre, mio padre e mio fratello.
A tutti coloro che mi vogliono bene.
Con animo aperto al nuovo e perchè no anche a un po' d'avventura!
A Picus.

Acknowledgements

Intendo ringraziare il corpo docenti di Data Science che ci ha messo il cuore (e le giuste competenze) nel portare avanti questo nuovo progetto di laurea magistrale. Ringrazio il DLR per avermi concesso l'occasione di un tirocinio all'estero in un gruppo di lavoro di altissimo profilo tecnico-scientifico. Ringrazio la mia famiglia senza la quale non avrei potuto mai fare nulla. Ringrazio i *"Lumi Viventi di Misericordia"* con Padre Franco e Graziella che mi hanno accompagnato sempre durante tutto il mio cammino.

Contents

"Voi occidentali, avete l'ora ma non avete mai il tempo."

Gandhi

Chapter 1

Introduction

Time series are a particular kind of data made of variables that are observed in different timestamps. Multivariate time series consists of more than one variable at each timestamp; univariate time series means that at each timestamp you observe only one variable. The focus of the work was implementing models that predict variables along future timestamps or finding not expected sequences in the series and label it as anomalies. Models used try to learn correlation along the time dimension and among variables, so they can learn hidden schemes from normal sequences and reproduce it to forecast variables. An unsupervised method is then used to automatically assess hundreds to thousands of diverse streams and determine whether resulting prediction errors represent anomalies. Lastly, strategies for mitigating false positive anomalies are outlined and are a key element in developing user trust and improving utility in a production system. Consider a time series $X = x^1, x^2, ..., x^n$ where each step $x^t \in R^m$ in the time series is an m-dimensional vector $x_1^t, x_2^t, ..., x_m^t$, whose elements correspond to input variables. A sequence length l_s determines the number of points to input into the model for prediction. A prediction length l_p then determines the number of steps ahead to predict, where the number of dimensions d being predicted is $1 \leq d \leq m$. Univariate time series correspond to $d = 1$, Multivariate time series correspond to $d > 1$. Temporal data is common in data mining applications. Typically, this is a result of continuously occurring processes in which the data is collected by hardware or software monitoring devices. The diversity of domains is quite significant and extends from the medical to the financial domain. Some examples of such data are as follows:

- *Sensor data*: Sensor data is often collected by a wide variety of hardware and other monitoring devices. Typically, this data contains continuous readings about the underlying data objects. For example, environmental data is commonly collected with different kinds of sensors that measure temperature, pressure, humidity, and so on. Sensor data is the most common form of time series data.

- *Medical devices*: Many medical devices such as electrocardiogram (ECG) and electroencephalogram (EEG) produce continuous streams of time series data. These represent measurements of the functioning of the human body, such as the heart beat, pulse rate, blood pressure, etc. Real-time data is also collected from patients in intensive care units (ICU) to monitor their condition.

- *Financial market data*: Financial data, such as stock prices, is often temporal. Other forms of temporal data include commodity prices, industrial trends, and economic indicators.

The greater complexity of time series data enables a larger number of problem definitions. Most of the models can be categorized into one of two types:

- *Real-time analysis*: In real-time analysis, the data points in one or more series are analyzed in real time, to make predictions. Typically, a small window of recent history is used over the different data streams for the analysis. Examples of such analysis include forecasting, deviation detection, or event detection. When multiple series are available, they are typically analyzed in a temporally synchronized way. Even in cases where data mining applications such as clustering are applied to these problems, the analysis is typically performed in real time.

- *Retrospective analysis*: In retrospective analysis, the time series data is already available, and subsequently analyzed. The analysis of different time series within a database is sometimes not synchronized over time. For example, in a time series database of ECG readings, the data may have been recorded over different periods.

First of all it is important to explain what are the main components of a time series.

- **Long term trend** is the overall general direction of the data, obtained ignoring any short term effects such as seasonal variations or noise.

- **Seasonality**refers to periodic fluctuations that are repeated throughout all the time series period.
- **Stationarity** is an important characteristic of time series. A time series is said to be stationary if its mean, variance and covariance don't have significant changes over time. There are we many transformations that can extract the stationary part of a non-stationary process.
- **Noise** refers to random fluctuations or variations due to uncontrolled factors.
- **Autocorrelation** is the correlation between the time series and a lagged version of itself, and is used to identify seasonality and trend in time series data.

Given the growing availability of data and computing power in the recent years, Deep Learning has become a fundamental part of the new generation of Time Series Forecasting models, obtaining excellent results. While in classical statistical models - such as autoregressive models (AR) or exponential smoothing - feature engineering is performed manually and often some parameters are optimized also considering the domain knowledge, Deep Learning models learn features and dynamics only and directly from the data. Thanks to this, they speed up the process of data preparation and are able to learn more complex data patterns in a more complete way. Before speaking about Deep Learning methods for Time Series Forecasting, it is useful to recall that the most classical statistical models used to solve this problem are ARIMA models and exponential smoothing. ARIMA stands for combination of Autoregressive (AR) and Moving Average (MA) approaches within building a composite model of the time series. This model is very simple, but might have good results. It includes parameters to account for seasonality, long term trend, autoregressive and moving average terms, in order to handle the autocorrelation embedded in the data. In Exponential smoothing forecasts are made on the basis of weighted averages like in ARIMA models, but in this case different decreasing weights are assigned to each observations and less importance is given to observations as we move further from the present. It is well known that these traditional statistical models have many limitations:

- they are not able to recognize complex patterns in the data;

- they usually work well only in few-steps forecasts, not in long term forecast.

Regarding deep learning models, the use of recurrent networks is known in scientific literature, in particular the lstm represent an evolution of them that solves gradient vanishing problems. More advanced architectures followed one another such as encoder-decoders that it could be considered as a block box that maps an input sequence to an output sequence, attention mechanisms that consider which input data affect the outputs the most and transformers that are well suited to sequential data. A very famous task accomplished in time series problems concerns forecasting. Time series forecasting involves taking models fit on historical data (the training set) and using them to predict future observations (the test set). At the first step past observations are collected and analyzed to develop a suitable mathematical model which captures the underlying data generating process for the series. In the second step the future events are predicted using the model. This approach is particularly useful when there is a lack of a satisfactory explanatory model. Making predictions about the future is called extrapolation in the classical statistical handling of time series data. More modern fields focus on the topic and refer to it as time series forecasting. The skill of a time series forecasting model is determined by its future prediction performance. Time series forecasting has important applications in various fields. Over the past several decades many efforts have been made by researchers for the development and improvement of suitable time series forecasting models. This is often at the expense of being able to explain why a specific prediction was made, confidence intervals and even better understanding the underlying causes behind the problem. An other important task accomplished with time series data concerns the outlier detection that consists of finding anomaly patterns in the sequence that can represent fault events. The goal of the following work is to describe the *Neural Sequence Analysis Toolbox* that allows the user to treat multivariate time series, by pre-processing them and searching anomaly patterns in the sequences using Deep learning methodologies. It is possible also to forecast the variables in the future timestamps.

Chapter 2

Preprocessing

2.1 Introduction

It is often beneficial to preprocess the data before feeding it to the models. Think of variables with different scales, or even when they have the same scale, small ranges of variables close to zero are beneficial for the optimization processes of neural networks. Sometimes in data such as time series noise can be found that should be removed. Hence the need to offer the user the possibility to perform data preprocessing in our toolbox.

2.2 Standardization & Normalization

Normalizing a vector most often means dividing by a norm of the vector. It also often refers to rescaling by the minimum and range of the vector, to make all the elements lie between 0 and 1 thus bringing all the values of numeric columns in the dataset to a common scale. The normalization implemented in the toolbox is:

$$\frac{x - x_{min}}{x_{max} - x_{min}}$$

$x \rightarrow is\ the\ variable$
Standardizing a vector most often means subtracting a measure of location and dividing by a measure of scale. For example, if the vector contains random values with a Gaussian distribution, you might subtract the mean

and divide by the standard deviation, thereby obtaining a "standard normal" random variable with mean 0 and standard deviation 1. The mathematical formulation of this operation is:

$$\frac{x - \mu}{\sigma}$$

$x \rightarrow is\ the\ variable$
$\mu \rightarrow is\ the\ expected\ values\ of\ the\ variable$
$\sigma \rightarrow is\ the\ square\ root\ of\ the\ variance\ of\ the\ variable$

2.3 B-Splines

Splines are a very useful tool for smoothing time series. In case there is some noise in the series, smoothing the signals can be considered a good preprocessing strategy. The idea of B-splines as explained in [7] is to construct basis functions and rewrite the original signal as a sum of the basis functions multiplied with suitable coefficients obtaining in this way an approximation of the original signal.

$$\text{approximation's signal} = \sum_{i}^{n} \lambda_i f_b^i$$

$f_b^i \rightarrow is\ the\ i\ basis\ functions$
$\lambda_i \rightarrow is\ the\ i\ coefficient$
$n \rightarrow is\ the\ number\ of\ basis\ functions$

The λ coefficients are estimated with a linear regression using as predictors the basis functions. In the following we describe in details the construction of the basis functions. Define an augmented knot sequence:

$$\xi = (\xi_{-p}, ..., \xi_0, \boldsymbol{\xi}, \xi_{k+1}, ..., \xi_{k+p+1})$$

$p \rightarrow is\ the\ degree\ of\ the\ spline$
$k \rightarrow is\ the\ number\ of\ interior\ knots$
for $i = -p, ..., K + p$, let

$$B_{i,0}(x) = \begin{cases} 1 \text{ x } \subset |\xi_i, \xi_{i+1}| \\ 0 \text{ otherwise} \end{cases}$$

where by convention, $B_{i,0}(x) = 0$ if $\xi_i = \xi_{i+1}$
the i^{th} B-spline basis function of degree j, j = 1, . . ., p, is given by:

$$B_{i,j}(x) = \frac{x - \xi_i}{\xi_{i+j} - \xi_i} B_{i,j-1}(x) + \frac{-x + \xi_{i+j+1}}{\xi_{i+j+1} - \xi_{i+1}} B_{i+1,j-1}(x)$$

for $i = -p, ..., k + p - j$
In Figure 2.1 you can see an example of basis functions of degree = 3.

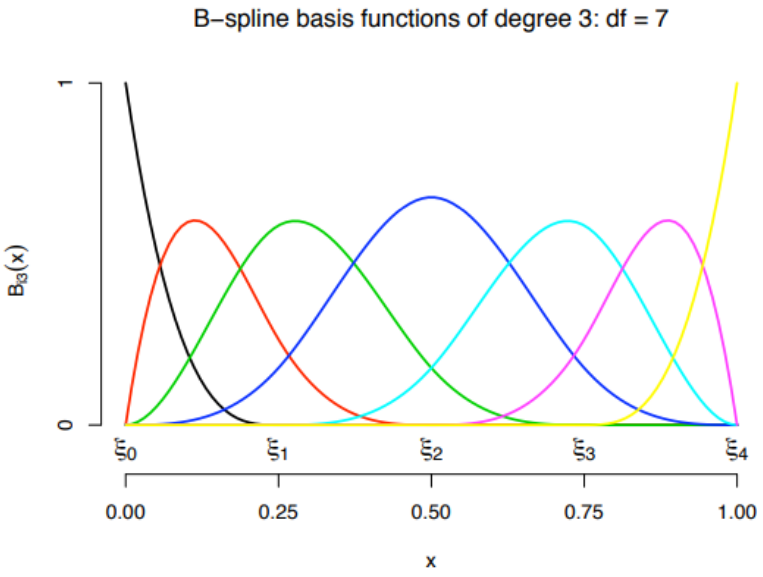

Figure 2.1: Basis functions.

2.4 Wavelets

We can use the Fourier Transform to transform a signal from its time-domain to its frequency domain. The peaks in the frequency spectrum indicate the most occurring frequencies in the signal. The larger and sharper a peak is, the more prevalent a frequency is in a signal. The location (frequency-value) and height (amplitude) of the peaks in the frequency spectrum then can be used as input for Classifiers like Random Forest or Gradient Boosting. The general rule is that this approach of using the Fourier Transform will

work very well when the frequency spectrum is *stationary*. That is, the frequencies present in the signal are not time-dependent; if a signal contains a frequency of x Hz this frequency should be present equally anywhere in the signal. The more non-stationary/dynamic a signal is, the worse the results will be. That's too bad, since most of the signals we see in real life are non-stationary in nature. Whether we are talking about ECG signals, the stock market, equipment or sensor data, etc, etc, in real life problems start to get interesting when we are dealing with dynamic systems. A much better approach for analyzing dynamic signals is to use the Wavelet Transform instead of the Fourier Transform. Fourier Transform works by multiplying a signal with a series of sine-waves with different frequencies we are able to determine which frequencies are present in a signal. If the dot-product between our signal and a sine wave of a certain frequency results in a large amplitude this means that there is a lot of overlap between the two signals, and our signal contains this specific frequency. This is of course because the dot product is a measure of how much two vectors / signals overlap. The weak side of the Fourier Transform is that it has a high resolution in the frequency-domain but zero resolution in the time-domain. This means that it can tell us exactly which frequencies are present in a signal, but not at which location in time these frequencies have occurred. In trying to overcome this problem, scientists have come up with the Short-Time Fourier Transform. In this approach the original signal is splitted into several parts of equal length (which may or may not have an overlap) by using a sliding window before applying the Fourier Transform. The idea is quite simple: if we split our signal into 10 parts, and the Fourier Transform detects a specific frequency in the second part, then we know for sure that this frequency has occurred between $\frac{2}{10}$th and $\frac{3}{10}$th of our original signal. The main problem with this approach is that you run into the theoretical limits of the Fourier Transform known as the uncertainty principle. The smaller we make the size of the window the more we will know about where a frequency has occurred in the signal, but less about the frequency value itself. The larger we make the size of the window the more we will know about the frequency value and less about the time. A better approach for analyzing signals with a dynamical frequency spectrum is the Wavelet Transform. The Wavelet Transform has a high resolution in both the frequency- and the time-domain. It does not only tell us which frequencies are present in a signal, but also at which time these frequencies have occurred. This is accomplished by working with different scales. First we look at the signal with a large scale/window and analyze

'large' features and then we look at the signal with smaller scales in order to analyze smaller features. In Figure 2.2 we can see the time and frequency

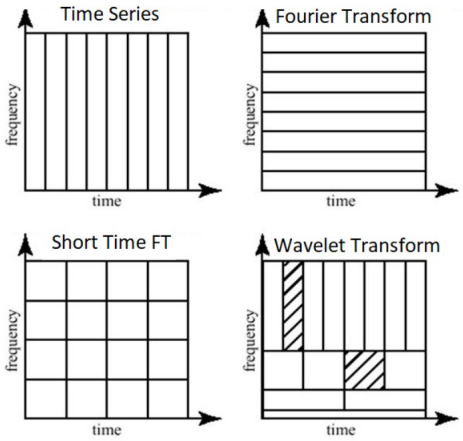

Figure 2.2: A schematic overview of the time and frequency resolutions of the different transformations in comparison with the original time-series data. The size and orientations of the block gives an indication of the resolution size.

resolutions of the different transformations. The size and orientation of the blocks indicate how small the features are that we can distinguish in the time and frequency domain. The original time-series has a high resolution in the time-domain and zero resolution in the frequency domain. This means that we can distinguish very small features in the time-domain and no features in the frequency domain. Opposite to that is the Fourier Transform, which has a high resolution in the frequency domain and zero resolution in the time-domain. The Short Time Fourier Transform has medium sized resolution in both the frequency and time domain. The Wavelet Transform has:

- for small frequency values a high resolution in the frequency domain, low resolution in the time- domain,
- for large frequency values a low resolution in the frequency domain, high resolution in the time domain.

In other words, the Wavelet Transforms makes a trade-off; at scales in which time-dependent features are interesting it has a high resolution in

the time-domain and at scales in which frequency-dependent features are interesting it has a high resolution in the frequency domain. The Fourier Transform uses a series of sine-waves with different frequencies to analyze a signal. That is, a signal is represented through a linear combination of sine-waves. The Wavelet Transform uses a series of functions called wavelets, each with a different scale. The word wavelet means a small wave, and this is exactly what a wavelet is. Since the Wavelet is localized in time, we can multiply our signal with the wavelet at different locations in time. We start with the beginning of our signal and slowly move the wavelet towards the end of the signal. This procedure is also known as a convolution. After we have done this for the original (mother) wavelet, we can scale it such that it becomes larger and repeat the process. So what is this dimension called scale? Since the term frequency is reserved for the Fourier Transform, the wavelet transform is usually expressed in scales instead. We can see that a higher scale-factor (longer wavelet) corresponds with a smaller frequency, so by scaling the wavelet in the time-domain we will analyze smaller frequencies (achieve a higher resolution) in the frequency domain. And vice versa, by using a smaller scale we have more detail in the time-domain. So scales are basically the inverse of the frequency. Another difference between the Fourier Transform and the Wavelet Transform is that there are many different families (types) of wavelets also called *mother wavelets*.

$$X_w(a,b) = \frac{1}{|a|^{0.5}} \int_{-\infty}^{\infty} x(t)\hat{\psi}(\frac{t-b}{a})dt$$

where $\psi(t)$ is the continuous mother wavelet which gets scaled by a factor of a and translated by a factor of b. The values of the scaling and translation factors are continuous, which means that there can be an infinite amount of wavelets. You can scale the mother wavelet with a factor of 1.3, or 1.31, and 1.311, and 1.3111 etc. When we are talking about the Discrete Wavelet Transform, the main difference is that the DWT uses discrete values for the scale and translation factor. The scale factor increases in powers of two, so a = 1, 2, 4, .. and the translation factor increases integer values (b = 1, 2, 3 ..). PS: The DWT is only discrete in the scale and translation domain, not in the time-domain. To be able to work with digital and discrete signals we also need to discretize our wavelet transforms in the time-domain. These forms of the wavelet transform are called the Discrete-Time Wavelet Transform and the Discrete-Time Continuous Wavelet Transform. In practice, the DWT is always implemented as a filter-bank. This means that it is implemented as

a cascade of high-pass and low-pass filters. This is because filter banks are a very efficient way of splitting a signal of into several frequency sub-bands. To apply the DWT on a signal, we start with the smallest scale. As we have seen before, small scales correspond with high frequencies. This means that we first analyze high frequency behavior. At the second stage, the scale increases with a factor of two (the frequency decreases with a factor of two), and we are analyzing behavior around half of the maximum frequency. At the third stage, the scale factor is four and we are analyzing frequency behavior around a quarter of the maximum frequency. And this goes on and on, until we have reached the maximum decomposition level. What do we mean with maximum decomposition level? To understand this we should also know that at each subsequent stage the number of samples in the signal is reduced with a factor of two. At lower frequency values, you will need less samples to satisfy the Nyquist rate so there is no need to keep the higher number of samples in the signal; it will only cause the transform to be computationally expensive. Due to this downsampling, at some stage in the process the number of samples in our signal will become smaller than the length of the wavelet filter and we will have reached the maximum decomposition level. To give an example, suppose we have a signal with frequencies up to 1000 Hz. In the first stage we split our signal into a low-frequency part and a high-frequency part, i.e. 0-500 Hz and 500-1000 Hz. At the second stage we take the low-frequency part and again split it into two parts: 0-250 Hz and 250-500 Hz. At the third stage we split the 0-250 Hz part into a 0-125 Hz part and a 125-250 Hz part. This goes on until we have reached the level of refinement we need or until we run out of samples. We can easily visualize in Figure 2.3 when we apply the DWT on a chirp signal. A chirp signal is a signal with a dynamic frequency spectrum; the frequency spectrum increases with time. The start of the signal contains low frequency values and the end of the signal contains the high frequencies. This makes it easy for us to visualize which part of the frequency spectrum is filtered out by simply looking at the time-axis. In our work in the preprocessing phase wavelets are used to remove noise from signal simply removing high frequencies from the original signal setting to 0 detail coefficients related to high frequencies.

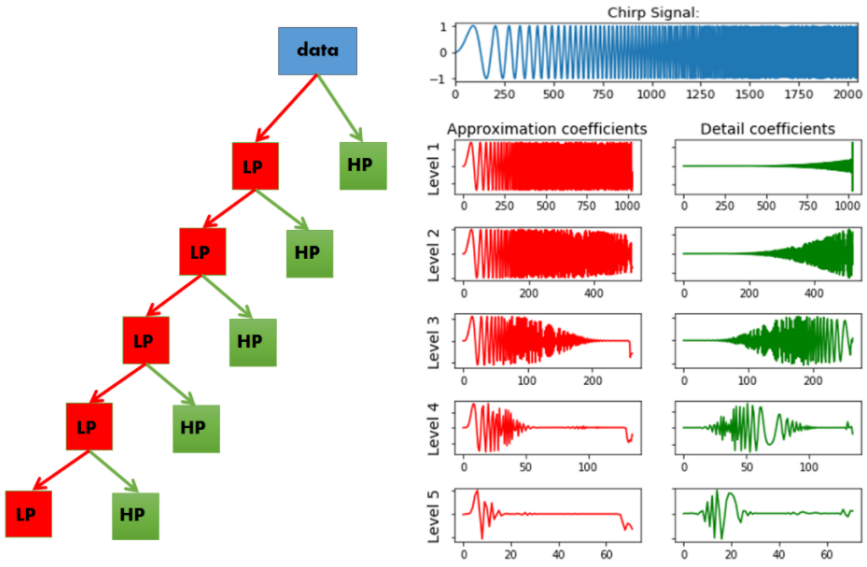

Figure 2.3: he approximation and detail coefficients of the sym5 wavelet (level 1 to 5) applied on a chirp signal, from level 1 to 5. On the left we can see a schematic representation of the high pass and low pass filters applied on the signal at each level.

Chapter 3

Outlier Detection and Forecasting

3.1 Introduction

Possible applications of time series analysis are outlier detection and forecasting. By outlier detection we mean the search for anomalous pattern in the data. It is useful to consider three categories of anomalies: *point*, *contextual*, and *collective*. *Point* anomalies are single values that fall within low-density regions of values, *collective* anomalies indicate that a sequence of values is anomalous rather than any single value by itself, and *contextual* anomalies are single values that do not fall within low-density regions yet are anomalous with regard to local values. The strategy used in this work to accomplish this tasks are Error Based Method and GANs. By forecasting we mean the ability of the model to learn the behavior of the sequence in the past and predict its behavior in the future, repeating patterns seen. Neural networks are capable of grasping correlations along the temporal direction and reproposing them in subsequent timestamps.

3.2 Error Based Method

By Error Based Method we mean a class of algorithm that accomplish outlier detection for time series performing predictions in future timesteps and evaluating the differences among forecasting and true values. After choosing a threshold all points whose errors overcome this threshold are considered anomalies. In the present work the unsupervised method "Dynamic Thresh-

olding" that is explained in [4] is used to automatize the research of the threshold avoiding statistical assumptions on the differences among true values and predicted values. An other effort was accomplished to mitigate false positive using "Pruning" procedure explained in [4]; it is important to develop reliable systems. Te following models are deep neural networks that perform forecasting.

3.2.1 Encoder-Decoder

Introduced for the first time in 2014 by Google with the paper [3], a sequence to sequence model aims to map a fixed-length input with a fixed-length output where the length of the input and output may differ.

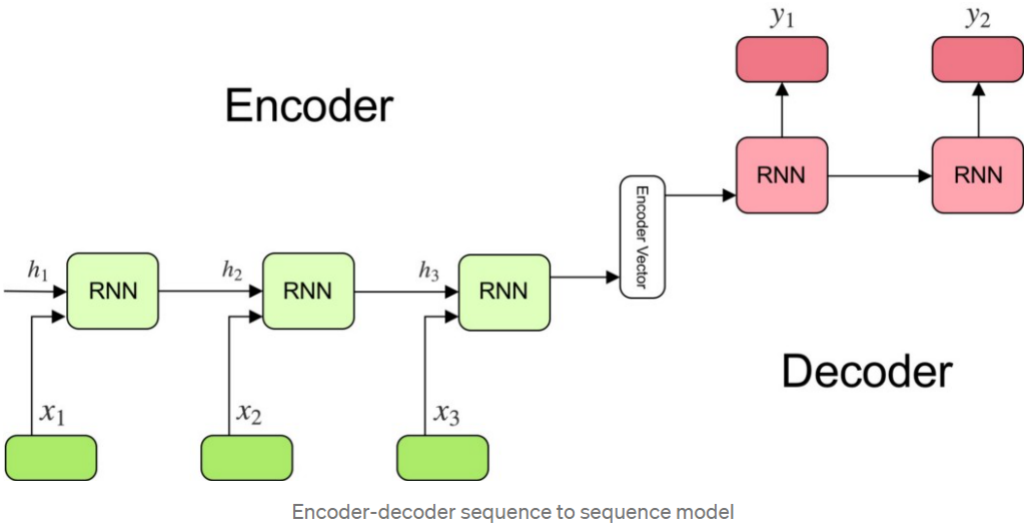

Figure 3.1: Sequence to sequence learnig.

As you can see the model consists of 3 parts:

- *Encoder:* A stack of several recurrent units (LSTM or GRU cells for better performance) where each accepts a single element of the input sequence, collects information for that element and propagates it forward.

- *Encoder Vector:* This vector aims to encapsulate the information for all input elements in order to help the decoder make accurate predictions.
- *Decoder:* A stack of several recurrent units where each predicts an output y_t at a time step t.

The core of the Encoder and Decoder is the LSTM network. LSTM stands for Long Short Term Memory [5]. LSTM can be used to solve problems faced by the RNN model. So, it can be used to solve:

- Long term dependency problem in RNNs.
- Vanishing Gradient Exploding Gradient.

The heart of a LSTM network is it's cell or say cell state which provides a bit of memory to the LSTM so it can remember the past. In LSTM we will

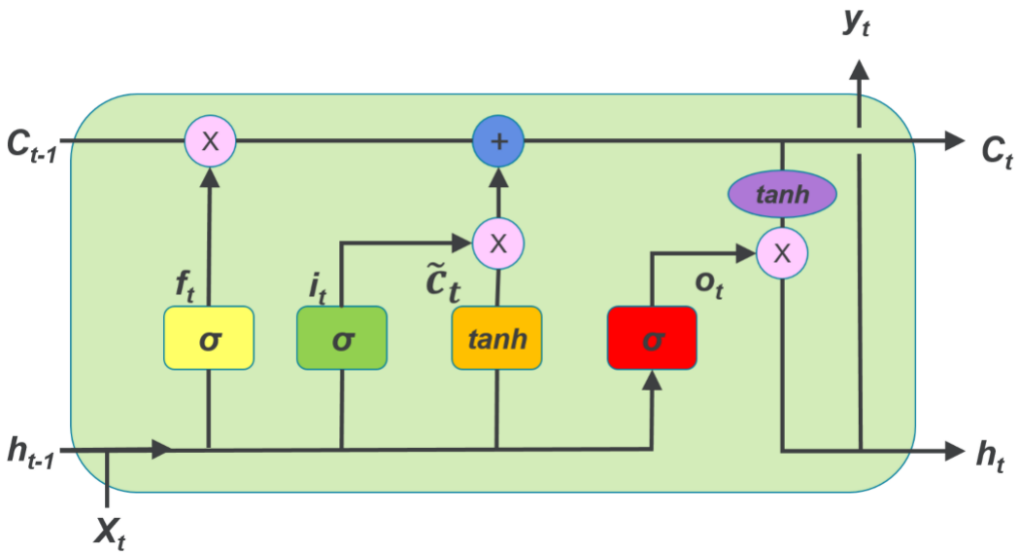

Figure 3.2: Long Short Term Memory.

have 3 gates:

- Input Gate.
- Forget Gate.

- Output Gate.

Input Gate tells us that what new information we're going to store in the cell state. Forget gate tells the information to throw away from the cell state. Output gate is used to provide the activation to the final output of the lstm block at timestamp 't'. The following are the equations that describe the behavior of the input, forget and output gates:

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

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

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

$i_t \rightarrow inputgate$
$f_t \rightarrow forgetgate$
$o_t \rightarrow ouputgate$
$w_x \rightarrow weight\ of\ the\ respective\ gate(x)$
$h_{t-1} \rightarrow output\ of\ the\ previous\ lstm\ block$
$x_t \rightarrow input\ at\ current\ timestamp$
$b_x \rightarrow biases\ of\ respective\ gate(x)$

The following equations describe the cell state, the candidate cell state and the final output:

$$\tilde{c}_t = tanh(w_c[h_{t-1}, x_t] + b_c)$$

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$$

$$h_t = o_t * tanh(c_t)$$

$c_t \rightarrow cellstate\ (memory)\ at\ timestamp\ t$
$\tilde{c}_t \rightarrow represents\ candidate\ for\ cellstate\ at\ timestamp\ t$

Now, from the above equation we can see that at any timestamp, our cell state knows that what it needs to forget from the previous state $f_t * c_{t-1}$ and what it needs to consider from the current timestamp $i_t * \tilde{c}_t$. note: * represents the element wise multiplication of the vectors. Lastly, we filter the cell

state and then it is passed through the activation function which predicts what portion should appear as the output of current lstm unit at timestamp t. We can pass this h_t the output from current lstm block through a dense layer to get the predicted output y_t from the current block.

3.2.2 Attention Model

In the paper *Neural Machine Translation by jointly learning to align and translate*, it was conjectured that the use of a fixed-length vector is a bottleneck in improving the performance of this basic encoder–decoder architecture, and it was proposed to extend this by allowing a model to automatically (soft-)search for parts of a source sequence that are relevant to predicting a particular future timestamp, without having to form these parts as a hard segment explicitly. A potential issue with the encoder–decoder approach is that a neural network needs to be able to compress all the necessary information of a source sentence into a fixed-length vector. This may make it difficult for the neural network to cope with long sequences resulting in degrading predictions. To overcome this obstacle with attention mechanism the decoder decides parts of the source sequence to pay attention to. The

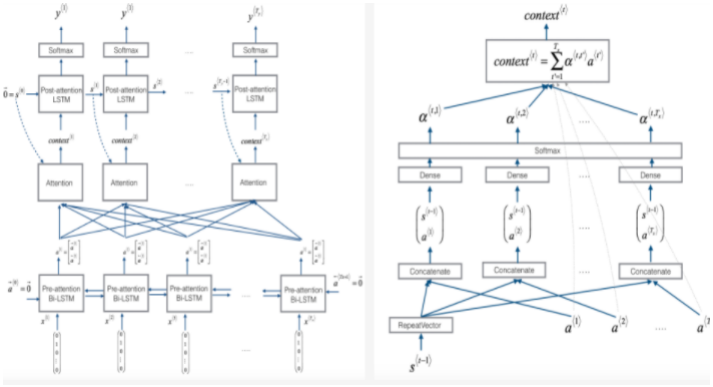

Figure 3.3: Attention Mechanism.

context vector c_i showed in Figure 3.3 is computed as below:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

$h_j \rightarrow is\ the\ hidden\ state\ of\ the\ encoder\ .\ In\ Figure\ 3.3\ is\ referred\ as\ a$
The weight α_{ij} is computed by:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

where

$$e_{ij} = a(s_i, h_j)$$

$s_i \rightarrow is\ the\ hidden\ state\ of\ the\ decoder$

The alignment model directly computes a soft alignment, which allows the gradient of the cost function to be backpropagated through. This gradient can be used to train the alignment model as well as the whole model jointly. As you notice in Figure 3.3 the encoder consists of a Bidirectional LSTM. The usual RNN, reads an input sequence x in order starting from the first symbol x_1 to the last one x_{Tx}. A BiRNN consists of forward and backward RNN's. The forward RNN $\overrightarrow{f}\ reads\ the\ input\ sequence\ as\ it\ is\ ordered (from\ x_1$ to $x_{Tx})$ and calculates a sequence of forward hidden states $\overrightarrow{h_1}, , \overrightarrow{h_{T_x}}$. The backward RNN $\overleftarrow{f}\ reads\ the\ sequence\ in\ the\ reverse\ order (from\ x_{Tx}$ to $x_1)$, resulting in a sequence of backward hidden states $\overleftarrow{h_1}, , \overleftarrow{h_{T_x}}$. We obtain an annotation for each timestamp x_j by concatenating the forward hidden state $\overrightarrow{h}j\ and\ the\ backward\ one\ \overleftarrow{h}j, i.e., h_j = [\overrightarrow{h_j}^T, \overleftarrow{h_j}^T]^T$. This sequence is used by the decoder and the alignment model later to compute the context vector.

3.2.3 Temporal Attention Model (or Multi Modal Attention)

Hori et al. in *"Attention-based multimodal fusion for video description"* proposed to handle multimodal data by fusing features of different modalities such as texts, audios and videos together with softly assigned weights of each modality. Fan et al. in *"Multi Horizon Time Series Forecasting with Temporal Attention Learning"* treat different periods of the history as different modalities and they combine them by learning relative importance of each period for predicting current time step. The idea is to divide the input sequence in multiple periods , apply attention mechanism for each period and reapply attention mechanism in a hierarchical way on the result of the precedent step. The Figure 3.4 below will clarify the concept: The following are

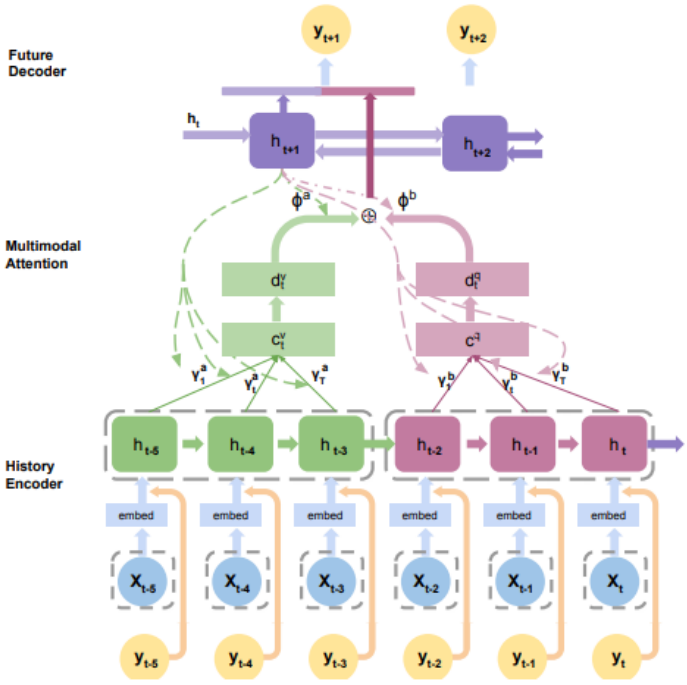

Figure 3.4: Temporal Attention

the equations that describe the model:

Attention

$$d_t = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

$h_j \rightarrow$ *is the hidden state of the encoder . In Figure* 3.3 *is referred as a* The weight α_{ij} is computed by:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

where

$$e_{ij} = a(s_i, h_j)$$

$s_i \rightarrow$ *is the hidden state of the decoder*

Multimodal Fusion

As shown in Fig 3.4, we apply temporal attention mechanism on M periods of historical data (M=2 in Fig 3), and fuse them with the multimodal attention weights $\phi_t^{1\cdots M}$ obtained by interacting the previous hidden state s_{t-1} with the transformed content vectors d_t^m:

$$c_t = \sum_{m=1}^{M} \phi_t^m d_t^m$$

$$\phi_t^m = \frac{\exp(e_{tm})}{\sum_{k=1}^{M} \exp(e_{tk})}$$

$$e_t^m = a(s_t, d_t^m)$$

It is important to outline that in the toolbox implemented the architecture is a 2 modal attention model (M = 2) and in the configuration file *input_configuration* \ *inputOutDet* to set the input window and the length of one period you have to use *iwindow* and *first_period* i.e: *iwindow*=200 and *first_period*=100 means input sequence equal to 200 with 2 periods each of length 100.

3.2.4 Dynamic Thresholding

It is required a fast, general, and unsupervised approach for determining if predicted values are anomalous. One common approach is to make Gaussian assumptions about the distributions of errors. However, this approach often becomes problematic when parametric assumptions are violated and in the paper *"Detecting Spacecraft Anomalies Using LSTM and Nonparametric Dynamic Thresholding"* it is proposed an approach that efficiently identifies extreme values without making such assumptions. Once a predicted value $\hat{y}^t$ is generated for each step t, the prediction error is calculated as $e_t = y^t - \hat{y}^t$ where y^t is the true value and $\hat{y}^t$ is the predicted value. Each e_t is appended to a one-dimensional vector of errors: $\mathbf{e}_s = e_s^{t-h}, ..., e_s^{t-1}, e_s^t$ where h is the number of historical error values used to evaluate current errors. In our toolbox h is equal to the number of points of test data. The set of errors $\mathbf{e}$ are then smoothed to dampen spikes in errors that frequently occur with LSTM-based predictions – abrupt changes in values are often not perfectly predicted and result in sharp spikes in error values even when this behavior

is normal. It is possible using the configuration parameters *smooth* that you find in the file *input_configuration* *inputOutDet*. To evaluate whether values are nominal, it is set a threshold for prediction errors – values corresponding to errors above the threshold are classified as anomalies. It is proposed an unsupervised method that achieves high performance with low overhead and without the use of labeled data or statistical assumptions about errors. With a threshold ϵ selected from the set:

$$\epsilon = \mu(\mathbf{e}_s) + \mathbf{z}\sigma(\mathbf{e}_s)$$

Where ϵ is determined by maximzing the following quantity:

$$\frac{\Delta\mu(e_s)/\mu(e_s) + \Delta\sigma(e_s)/\sigma(e_s)}{|e_a| + |E_{seq}|^2}$$

Such that:

$$\Delta\mu(e_s) = \mu(e_s) - \mu(e_s|e_s < \epsilon)$$

$$\Delta\sigma(e_s) = \sigma(e_s) - \sigma(e_s|e_s < \epsilon)$$

$$e_a = \{e_s|e_s > \epsilon\}$$

$$E_{seq} = continuos\ sequences\ of\ e_a$$

Values evaluated for ϵ are determined using z where z is an ordered set of positive values representing the number of standard deviations above $\mu(e_s)$. Values for z depend on context, and you can set it in our toolbox using the configuration parameters *z_min* and *z_max* in the file *input_configuration* *inputOutDet*. In simple terms, a threshold is found that, if all values above are removed, would cause the greatest percent decrease in the mean and standard deviation of the smoothed errors $\boldsymbol{e}_s$. The function also penalizes for having larger numbers of anomalous values $|e_a|$ and sequences $|E_{seq}|$ to prevent overly greedy behavior.

3.2.5 Pruning

The precision of prediction-based anomaly detection approaches heavily depends on the amount of historical data h used to set thresholds and make judgments about current prediction errors. At large scales it becomes expensive to query and process historical data in real-time scenarios and a lack of history can lead to false positives that are only deemed anomalous

because of the narrow context in which they are evaluated. Additionally, when extremely high volumes of data are being processed a low false positive rate can still overwhelm human reviewers charged with evaluating potentially anomalous events. To mitigate false positives and limit memory and compute cost, in the paper *"Detecting Spacecraft Anomalies Using LSTM and Nonparametric Dynamic Thresholding"* it is introduced a pruning procedure in which a new set, $\boldsymbol{e}_{max}$, is created containing $\max(E_{seq})$ for all sequences sorted in descending order. It is also added the maximum error that isn't anomalous. The sequence is then stepped through incrementally and the percent decrease $\frac{e_{max}^{i-1} - e_{max}^{i}}{e_{max}^{i-1}}$. If at some step i a minimum percentage decrease p is exceeded by d_i, all $e_{max}^{j} | (j < i)$ and their corresponding anomaly sequences remain anomalies. If the minimum decrease p is not met by d_i and for all subsequent errors, those error sequences are reclassified as nominal. This pruning helps ensures anomalous sequences are not the result of regular noise within a stream, and it is enabled through the initial identification of sequences of anomalous values via thresholding. The Figure 3.5 clarifies in a visual way the concept. In the file *input_configuration \inputOutDet* the pruning procedure can be controlled using the p parameter.

3.3 GANs

An alternative method to accomplish outlier detection in addition to error based is to use the GANs in which the model learns to reproduce the distribution of points and is able to understand when a point is not part of the underlying distribution and is therefore an anomaly. In the paper "Generative adversarial Nets" it was proposed a new framework for estimating generative models via an adversarial process, in which the authors simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to 1/2 everywhere. In the adversarial nets framework, the generative model is pitted against an adversary: a discriminative model that learns to determine whether a sample is from the model distribution or the data distribution. The generative model can be thought of as analogous to

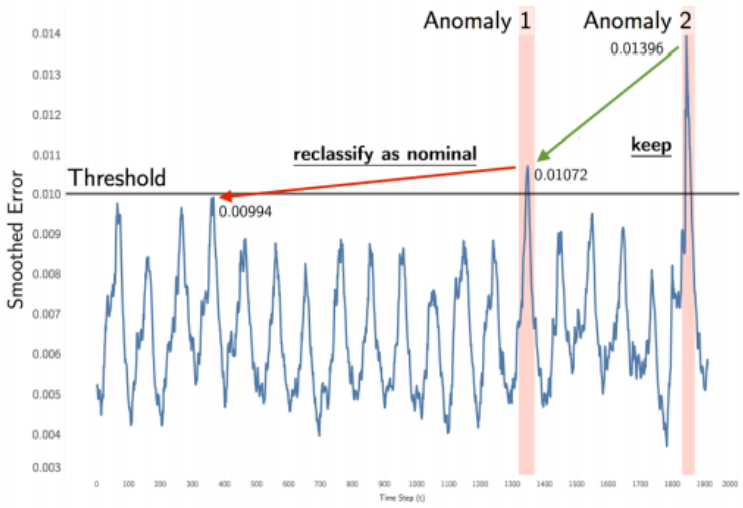

Figure 3.5: **Pruning procedure**
This example demonstrates the anomaly pruning process. In this scenario $\boldsymbol{e}_{max} = [0.01396, 0.01072, 0.00994]$ and the minimum percent decrease p = 0.1. The decrease from Anomaly 2 to Anomaly 1 $d_1 = 0.23 > p$ and this sequence retains its classification as anomalous. From Anomaly 1 to the next highest smoothed error ($e_s = 0.0099$) $d_2 = .07 < p$ so this sequence is reclassified as nominal.

a team of counterfeiters, trying to produce fake currency and use it without detection, while the discriminative model is analogous to the police, trying to detect the counterfeit currency. Competition in this game drives both teams to improve their methods until the counterfeits are indistiguishable. To learn the generator's distribution p_g over data x, it is defined a prior on input noise variables $p_z(z)$ (*latent space*), then represent a mapping to data space as $G(z; \theta_g)$, where G is a differentiable function represented in this toolbox by a convolutional neural network with parameters θ_g. It is also defined a second convolution neural network $D(x; \theta_d)$ that outputs a single scalar. D(x) represents the probability that x came from the data rather than p_g. We train D to maximize the probability of assigning the correct label to both training examples and samples from G. We simultaneously train G to minimize lg(1-$D(G(z))$): In other words, D and G play the following two-player

minimax game with value function V(G, D):

$$min_G max_D V(D,G) = E_{x \sim p_{data(x)}}[\lg D(x)] + E_{z \sim p_z(z)}[\lg(1\text{-}D(G(z)))].$$

We alternate between k steps of optimizing D and one step of optimizing G. The Figure 3.6 shows the training process. After the training process the

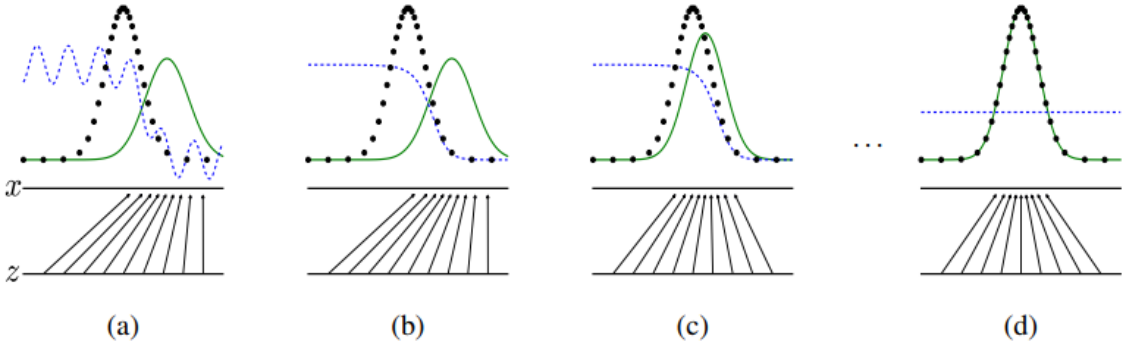

Figure 3.6: Generative adversarial nets are trained by simultaneously updating the discriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) p_x from those of the generative distribution p_g(G) (green, solid line). The lower horizontal line is the domain from which z is sampled, in this case uniformly. The horizontal line above is part of the domain of x. The upward arrows show how the mapping x = G(z) imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a)(b) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (c) After an update to G, gradient of D has guided G(z) to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{data}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(x) = 1/2$

discriminator is able to recognize points that come from real distribution or not (anomalies). Consider that the core network of the GANs used in this toolbox is a convolutional neural network. The multivariate time series input is converted in a frequence-time domain (Scaleogram) using wavelet and

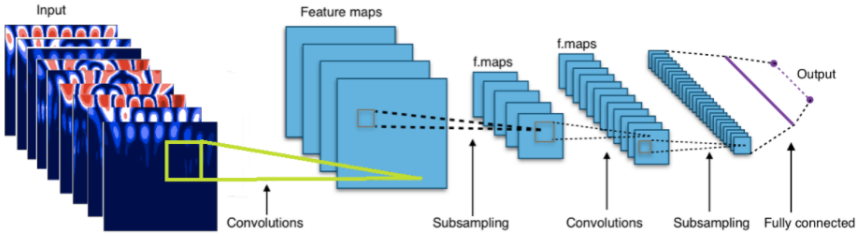

Figure 3.7: Each 2D image is the application of a DWT to each univariate time series that result in a scaleogram. We collect the scaleograms binding them on the third direction. After we apply the convolutional neural network that is the core of the generator and of the discriminator.

become the input of the model as showed in Figure 3.7. The scaleogram can not only be used to better understand the dynamical behavior of a system, but it can also be used to distinguish different types of signals produced by a system from each other. The strategy used is to place the scaleograms of each signal on top of each other and create one single image with different channels. In the file *input_configuration* *inputOutDet* you can control the dimension of the latent space using *latent_dim* and also the number of filters of the convolutional block using *n_filter_discriminator* and *n_filter_generator*.

3.4 Forecasting

Time series forecasting problem is to study how to predict the future accurately based on the historical observations. Increasing forecasting accuracy is beneficial to operational efficiency in many aspects of society. For example, data-driven demand forecasting techniques enable online retailers to gain a better understanding of market demands, thus improving supply chain operation efficiency such as delivery speed and product in-stock rate. In the above sections are already described Encoder-Decoder, Attention and Temporal-Attention that allow forecasting and can be used for anomaly detection also thanks to the Error-Based-Method. In the following section it will be described a model that is not known in literature that automatically decide how many future predictions can be considered reliable and the timestamps over are set to 0. This new model is based on a mask that leaves unaltered

predictions considered reliable and sets unreliable ones to 0 and can be integrated with previous models simply by adding the mask that will depend on a parameter that will be learned in the backpropagation phase. This model cannot be used for anomaly detection purpose but only for forecasting application.

3.4.1 Variable-Horizon Model

As already said above, it is important to make long-term forecasts, however one wonders how reliable the forecasts are. An original idea developed in the following work is to introduce a mask that set to 0 the predictions not considered reliable and the number of reliable predictions is learned from the data. We have to set a *max_horizon* that is a int number that specify the maximum horizon in the future that we would hope to predict. With m we refer to the number of reliable timestamps in the future, as decimal of *max_horizon*, that the model will predict. The simple idea is to introduce a mask in the loss function, as you can observe in Figure 3.8, that depend to a parameter that has to be learned in the training phase. In this way the length of the mask and as a consequence the length of the predictions is learned from data.

$$L = E(v_a, v_p) * f_{mask}(m)$$

$$f_{mask} = 0.5 * (1 + tanh((m * h_{max} - \hat{h})/\epsilon))$$

$m \rightarrow number\ of\ reliable\ points\ as\ decimal\ of\ the\ maximum\ horizon$
$\hat{h} \rightarrow (1, 2, ..., h_{max})\ with\ h\_max\ maximum\ horizon$
It is important to outline that m is a number among 0 and 1 and it is useful for the training phase: optimize parameter in this range is easier.

3.5 Hyperparameter optimization

In machine learning, hyperparameter optimization or tuning is the problem of choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a parameter whose value is used to control the learning process. By contrast, the values of other parameters (typically node weights) are learned. The same kind of machine learning model can require different constraints, weights or learning rates to generalize different data patterns. These measures are called hyperparameters, and have to be tuned so that the

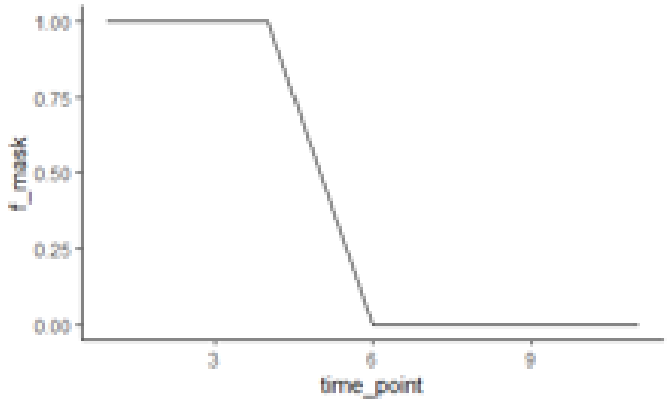

Figure 3.8: As you can observe the initial points are not masked meanwhile the points that are not considered reliable are set to 0.

model can optimally solve the machine learning problem. Hyperparameter optimization finds a tuple of hyperparameters that yields an optimal model which minimizes a predefined loss function on given independent data.The objective function takes a tuple of hyperparameters and returns the associated loss.

3.5.1 Grid & Random Search

Grid Search is a search technique that has been widely used in many machine learning researches when it comes to hyperparameter optimization. Among other approaches to explore a search space, an interesting alternative is to rely on randomness by using the Random Search technique. The ideal problem that machine learning researchers would like to work on is the one admitting a convex objective function with no hyperparameters needed nor relaxations, yet powerful enough to provide the tiniest error on unseen data. However, hyperparameters are a necessary. An example of hyperparameter that you might have heard of is the learning rate in neural networks. Diving a little bit deeper, take regularized linear models as an example. A regularization term is incorporated into the objective function in order to enforce a determined penalty. Two well-known approaches are the imposition of the l_1-norm and l_2-norm on the parameter vectors to perform feature selection (Lasso) and control the complexity of the model (Ridge), respectively. However,penalities need to be optimized with a controlled parameter, which is

	1	2	3
20	(20,1)	(20,2)	(20,3)
60	(60,1)	(60,2)	(60,3)
80	(60,1)	(80,2)	(80,3)

Table 3.1: α is on the horizontal and β on the vertical columns.

called hyperparameter since it needs to be set before the training process. And for the mentioned examples, only one hyperparameter is necessary; usually, you would have to deal with more than one of them. Besides manually searching for good candidate values for hyperparameters, the most basic and straightforward approach for optimizing hyperparameters is the Grid Search (GS) technique. Basically, a list of candidate values for each hyperparameter is defined and evaluated. The name "grid" comes to the fact that all possible candidates within all needed hyperparameters are combined in a sort of grid. The combination yielding the best performance, preferably evaluated in a validation set, is then selected. As an example, suppose that α and β are hyperparameters that will be optimized using GS. Based on some hypothetical knowledge, we can guess that the candidates could be [1, 2, 3] and [20, 60, 80] for α and β, respectively. Thus, we can set up a grid of the values and their combinations in Table 3.1 From the grid we have just built, each combination is evaluated and the one yielding the best performance is selected. After this process, perhaps we could find out that (3, 60) was the best option for our problem. On the other hand, the global minimum could be located at (2.57, 58). This task, however, starts to become very time-consuming if there are many hyperparameters and the search space is huge. As an alternative to GS, one could rely on randomness through the Random Search (RS) technique. It means that among the all combinations of hyperprameters a random selection is performed to collect a subset of hyperparameters as it clarified in Figure 3.9. In our toolbox the random search algorithm is implemented and he parameter *percentage_to_eliminate* in *input_configuration* *inputOutDet* controls this random selection; *percentage_to_eliminate* equals to 0 corresponds to the grid search algorithm.

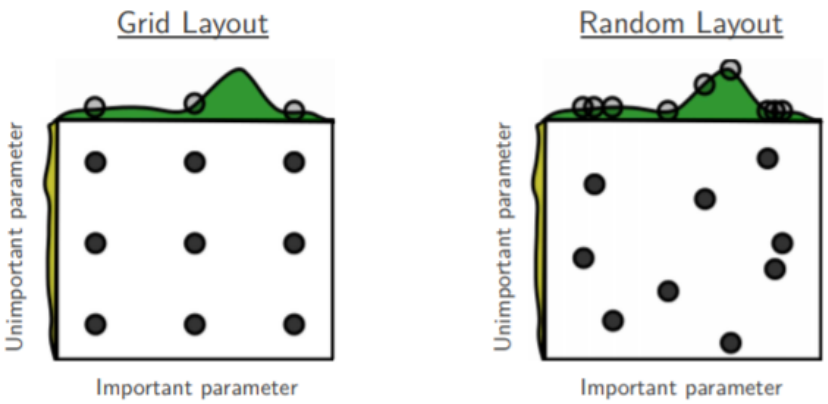

Figure 3.9: Search algorithms

Chapter 4

Experiments

4.1 Introduction

The experiments conducted to verify the performance of the models for outlier detection and forecasting applications are reported. The outlier detection experiments are conducted on public data of NASA and are compared the error based strategy and GANs to discover anomaly patterns in the sequences. The forecasting for each model will show the performances of the models with and without the variable horizon option on air pollution dataset.

4.2 OutlierDetection Experiments

Regarding the Error Based Strategy, models are trained to learn normal system behaviors using encoded command information and prior telemetry values. Predictions are generated at each time step and the errors in predictions represent deviations from expected behavior. It is then used a nonparametric, unsupervised approach for thresholding these errors and identifying anomalous sequences of errors. Concerning the GANs strategy, this framework is trained on normal data where the discriminator learns to distinguish data that come from normal or an anomaly distribution and after is used on test data. The raw data represent real spacecraft telemetry data and anomalies from the Soil Moisture Active Passive satellite (SMAP) and the Curiosity Rover on Mars (MSL). All data has been anonymized with regard to time and all telemetry values are pre-scaled between (-1,1) according to the min/max in

the test set. Channel IDs are also anonymized, but the first letter gives indicates the type of channel (P = power, R = radiation, etc.). Model input data also includes one-hot encoded information about commands that were sent or received by specific spacecraft modules in a given time window. No identifying information related to the timing or nature of commands is included in the data. Figure 4.1 shows the dataset used for experiments. You can find

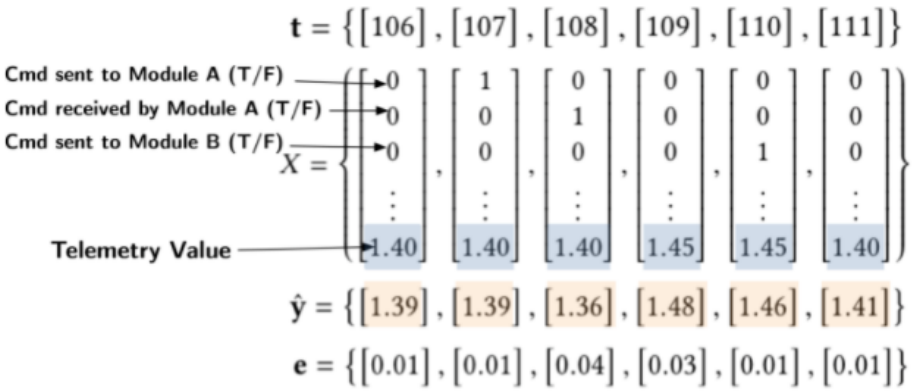

Figure 4.1: Multivariate time series. At the bottom you can see predictions and errors.

the data at the following link `https://github.com/khundman/telemanom`. Among the different channels available we performed experiments only on id channel "E-1" that presents contextual anomalies. For each model we report in Table 4.1 precision, recall and fscore. To remember the meaning of these metrics we report their mathematical formulation:

$$precision = \frac{t_p}{t_p + f_p}$$

$$recall = \frac{t_p}{t_p + f_n}$$

$$fscore = \frac{2 * precision * recall}{precision + recall}$$

$t_p \rightarrow$ *model labels as anomaly when it is a true anomaly*
$f_p \rightarrow$ *model labels as anomaly when it is not a true anomaly*
$f_n \rightarrow$ *model labels as normal when it is a true anomaly*

	precision	recall	fscore
Encoder-Decoder	0.57	0.81	0.67
Attention	0.41	0.84	0.55
Temporal-Attention	0.52	0.84	0.65
GANs	0.95	0.73	0.82

Table 4.1: Comparison among models

In the statistical analysis of binary classification, the F-1 score (also known as F-score or F-measure , literally "F measure") is a measure of the accuracy of a test . The measurement takes into account precision and recall of the test, where precision is the number of true positives divided by the number of all positive results so it takes in account the problem of false positive, while recall is the number of true positives divided by the number of all tests that should have been positive (i.e. true positives plus false negatives) so it takes in account all that situations in which the model doesn't see anomalies. The F 1 is calculated using the harmonic mean of precision and recall. It can assume values between 0 and 1. It assumes a value of 0 only if at least one of the two is equal to 0, while it assumes a value of 1 if both precision and recall are equal to 1.

As you can observe in Table 4.1 encoder decoder, attention and temporal attention are affected by the problem of false positives. Despite the pruning carried out to mitigate this phenomenon, the problem of false alarms remains. It is interesting to note that the GANs are not affected at all by this effect and even if they show a slightly lower recall than the models used with the error based strategy, they are the model with the highest performance if we look at the f-score index which it takes into account both the ability of GANs to detect anomalies and the problem of false positives. In Tab. 4.2 you can find the chosen hyperparameters. Notice that in erro based strategy the output window is equal to 1.

An aspect of the toolbox to improve is to consider the possibility of updating the model as new data is provided and to give alarms in real time.

4.3 Forecasting Experiments

Time series forecasting problem is to study how to predict the future accurately based on the historical observations. Models are trained on train

	inputwindow	neurons	epochs
Encoder-Decoder	250	80	35
Attention	125	100	35
Temporal-Attention	250	150	35
GANs	200	128*	1

Table 4.2: Hyperparameters obtained with Random Search Algorithm. *The neurons in GANs have to be thought as number of filters for convolution operation.

data and predictions are performed along timestamps of test data to accomplish a comparison among forecasted values and true values in order to compute performance metrics. The dataset contains 9360 instances of hourly averaged responses from an array of 5 metal oxide chemical sensors embedded in an Air Quality Chemical Multisensor Device. The device was located on the field in a significantly polluted area, at road level,within an Italian city. Data were recorded from March 2004 to February 2005 (one year)representing the longest freely available recordings of on field deployed air quality chemical sensor devices responses. Ground Truth hourly averaged concentrations for CO, Non Metanic Hydrocarbons (NMHC), Total Nitrogen Oxides (NOx), Nitrogen Dioxide (NO2) and indium oxide and were provided by a co-located reference certified analyzer. Carbon monoxide (CO) is of particular importance among the pollutants produced by combustion. It is a toxic, colorless, odorless, tasteless and non-irritating gas which, without adequate ventilation, can reach high concentrations. It is produced by incomplete combustion of any material organic, in the presence of low oxygen content in the environment. High concentrations reached in the body can be lethal. NMHC can be of natural and anthropogenic origin, are among the main pollutants emitted by petrochemical plants and refineries and can also be released during the drilling and extraction of crude oil. The acronym (NOx) generically identifies the nitrogen oxides that are produced as byproducts during a combustion that takes place using air. Nitrogen oxides, especially nitrogen dioxide are pollutants of the atmosphere and aggravate the conditions of the sick of asthma, children and those suffering from chronic respiratory disease or heart disease. Nitrogen dioxide (NO 2) is a red-brown, poisonous, pungent gas that smells similar to chlorine. Nitrogen dioxide has been produced on a large scale since 1908 and used for the production of

nitric acid . Nitrogen dioxide is produced in traces from oxygen and nitrogen as the two main components of the atmosphere during natural processes such as B. Indium(III) oxide (In2O3) is a chemical compound, an amphoteric oxide of indium. Since use as input all multivariate dataset resulted in worst performance we decided to consider uni-variate time series and we accomplished experiments with NMHC. You can find the data at the following link `https://archive.ics.uci.edu/ml/datasets/air+quality`. For each model the idea was to use the option of variable horizon using a maximum horizon equal to 100 and searching for the optimal future horizon. For each model (respectively Encoder-Decoder, Attention Mechanism, Temporal Attention) a grid from 10 to 100 has been built with 10 steps of different output windows on which to test the models without the "variable horizon" option. The idea is to check whether the optimal output time window predicted by the respective model with the variable horizon option actually matches the optimal window of the model used without using the above option. It is computed the mean absolute error (*mae*) on test data. The formulation of the *mae* is:

$$MAE = \sum_{j=1}^{n\_variables} \frac{1}{n_{variables}} \sum_{i=1}^{n_{time}} \frac{|y_i^j - x_i^j|}{n_{time}}$$

$y_i^j \rightarrow$ *is the prediction at timestamp i of variable j*
$x_i^j \rightarrow$ *is the observation at timestamp i of variable j*
$n\_time \rightarrow$ *is the number of timestamps*
$n\_variables \rightarrow$ *is the number of variables*

As you can see in Table 4.3 the experiments carried out with Encoder-Decoder show that the variable horizon option is very useful for finding the reliable time window on which to make predictions but due to the fact that the model has to learn this time window it results in more smoothed forecasts. The idea is to use variable horizon as a strategy to find an optimal output window and then run the original model on the found reliable future timestamps. As you can see in Table 4.4 and 4.5 the experiments carried out with Attention Mechanisms and Temporal Attention Mechanisms in conjunction with variable horizon option work very bad; instead of predicting an outputwindow of 60 and 10 respectively it predicts completely different outputwindows failing in its goal of finding the number of reliable predictions. We can observe from these experiments that the idea of Variable horizon of searching for the number of reliable predictions works quite good only for

	horizon	MAE
Encoder-Decoder	10	114
	20	153
	30	94
	40	102
	50	90
	60	120
	70	127
	80	115
	90	118
	100	91
Encoder-Decoder VH	42	237

Table 4.3: Performance of Encoder-Decoder on a grid of different output windows. The last row is the learned horizon of Encoder-Decoder variable horizon, as you can see it is in a region of a low MAE for Encoder-Decoder.

	horizon	MAE
Attention Mechanisms	10	114
	20	126
	30	125
	40	126
	50	109
	60	84
	70	235
	80	323
	90	244
	100	232
Attention Mechanisms VH	100	241

Table 4.4: Performance of Attention Mechanisms on a grid of different output windows. The last row is the learned horizon of Attention Mechanisms variable horizon, as you can see it works quite bad predicting an output window equal to 100 instead of 60 that is the optimal outputwindow.

	horizon	MAE
Temporal Attent. Mech.	10	82
	20	83
	30	120
	40	104
	50	178
	60	165
	70	239
	80	243
	90	268
	100	232
Temporal Attent. Mech. VH	58	222

Table 4.5: Performance of Temporal Attention Mechanisms on a grid of different output windows. The last row is the learned horizon of Attention Mechanisms variable horizon, as you can see it works quite bad predicting an output window equal to 58 that does not match the optimal outputwindow of 10.

Encoder-Decoder model. It seems that in the training process it is not able to learn in a proper way the number of reliable predictions simultaneously with the attention scores. The experiments were conducted on 25 epochs, 2 layers, 100 neurons, input window of 100, learning rate of 0.001 for models without the variable horizon option and 0.01 in the other circumstance.The initial value of the starting mask for variable horizon is 1.

Conclusions

The purpose of the following work was to explore time series methodologies starting from the first pre-processing pipelines that concern the possibility of bringing the different variables of the dataset to the same scale using standardization or normalization methods, up to the possibility of removing noise from data with smoothing techniques such as basis splines or removal of high frequencies thanks to wavelets. It is therefore essential to remove the noise from the data and have variables with comparable scales before feeding them to neural network algorithms. After this initial phase we explored different methodologies to accomplish anomaly detection tasks and forecasting tasks. By anomaly detection tasks we mean the capability to search pattern in the sequence that differ from the normal scheme of the sequence, meanwhile with forecasting task we mean the prediction of the behavior of the signal in the future timestamps. Regarding outlier detection, we compared two types of strategies: one called Error Based Method and the other based on GANs. The first is to make predictions of the sequence into the future and evaluate how far it is from the observed behavior. Since the predictions should give an idea of the normal behavior of the series since the model learns on data that does not have anomalies, if the difference between predictions and observed values exceeds a certain threshold implies the presence of an anomaly, otherwise it does not. The second strategy instead uses the capability of the discriminator of GANs to label sequences as anomaly or normal. We have observed from the experiments conducted on NASA public data (which are appropriately anonymized) that the models used with the logic of the error based method suffer from the problem of false positives despite strategies such as pruning have been applied to try to contain this phenomenon. GANs,

on the other hand, despite losing a little in the ability to detect anomalies, gain a lot in the problem of false positives and if you look at the f-score index as a metric, they turn out to be the best strategy for finding anomalies. Regarding forecasting, the experiments conducted with or without the variable horizon strategy (which allows to find the number of timestamps in the future on which the prediction is considered reliable) show that the attention mechanisms work very badly with the variable horizon strategy meanwhile this idea used in conjunction with the encoder-decoder model seems to lead to a solution for the output window which does not correspond to the global minimum but to a local minimum not far from the optimal solution. This means that you can use Encoder-Decoder plus Variable Horizon option to search a reliable number of future timestamps on which to forecast and after finding a reliable output window, you can retrain the Encoder-Decoder without Variable Horizon option using as output the number of timestamps that you found previously. With this work, a software has been produced that implements all the proposed solutions accompanied by a yaml configuration file on which all the parameters and the choices to be made for the processing of the time series can be set. In the appendix you will find the documentation. Regarding the software, there is more work to be done on cleaning the code, it is necessary to implement an optimization procedure of the hyperparameters also for the GANs (it is currently possible to use the random search only for encoder-decoder, attention and temporal attention). It must also be adapted for real time analysis and it needs a graphical interface too. Regarding the methodological perspective one limit of this work is that it is not implemented any procedure to construct confidence intervals and to deal with missing values. As future perspectives the purpose is to fill this gap and try to investigate graph neural networks too that seem to be a new frontier of research for time series.

Chapter A

Software Documentation

A.1 Introduction

This document is a reference to install and use the Neural Sequence Analysis Toolbox, a software that allows the user to develop complex studies on time series data including forecasting and outlier detection. Imagine having a dataset with different variables on the columns and with different measurements over time on the rows, for example having a set of data that has on the columns pressure and temperature and on the rows the different measures of them over time. Forecasting means taking selected variables as input for a certain number of timestamps and predicting their behavior (or the behavior of a subset of them) over a time window of a certain number of future timestamps. By outlier detection we mean the possibility of finding anomalous patterns, not expected, within the sequence of points. The network in the training phase learns to recognize only the sequences without anomalies and in the testing phase it will be presented with data in which there may be anomalies and being patterns that the network has never seen, so it will be able to recognize them as outliers.

A.2 Install

The following packages are required to run the code:

jupyter==1.0.0

numpy==1.19.5

pandas==1.2.4
Keras==2.4.3
tensorflow==2.4.1
matplotlib==3.4.1
seaborn==0.11.1
scikit-learn==0.24.2
PyYAML==5.4.1
scipy==1.6.3
PyWavelets==1.1.1
scikit-image==0.18.1
Python-version==3.7.10

A.3 Usage

To use the software it has to be launched "main.ipynb" from the root folder; before launching the software it is important to set well the input configuration. From the root folder it is possible to access to the "input_configuration". It is possible to choose among "inputForecast.yml" and "inputOutDet.yml". The first is the configuration file concerning the forecasting application, the second is related to the anomaly detection application. From the root you will find "results" folder where you will find forecasted values for forecasting application or anomaly labels(1 is anomaly, 0 is normal point) for outlier detection purpose. A detailed explanation of the software is given below.
N.B: the input files should be .npy files with variables along columns and timesteps along rows.

A.3.1 Forecasting application

As told in the introduction paragraph imagine having a dataset with different variables on the columns and with different measurements over time on the rows, forecasting means taking selected variables as input for a certain number of timestamps and predicting their behavior (or the behavior of a subset of them) over a time window of a certain number of future timestamps. It is important to configure parameters like number of input timestamps, number of output timestamps, variables to get in inputs, variables to forecast and other hyperparameters that define the neural models. It is possible thanks to the "inputForecast.yml" file that you find in the path

"root\input_configuration". The following is the description of the settings of the "inputForecast.yml". First of all it is important to set *Forecasting* equal to True and verify that *OutlierDetection* and *OutlierDetectionGANs* in "inputOutDet.yml" are settled to False. *is_variable_horizon* is a boolean, true if the choosen model is the variable horizon one. *architecture* is a string and can have the following values: Encoder-Decoder, Attention, MultiModalAttention, VariableHorizon. To know the details of the algorithms use the *algorithm's documentation* that you find in the root folder. Just a note on the variable horizon: it is not a standard model known in literature but it is a model that learns automatically how many timesteps for future prediction are reliable. The hyperparameter *layers* is an integer (not a list of integers, only one value is allowed!) and makes sense only for Encoder-Decoder. This is the number of encoder layers that is equal to the number of decoder layers. For the next settings of the architecture it is allowed to choose a list of values for each hyperparameter. An algorithm called Random Search will find the best configuration among all the possible configurations(cartesian product of all the values listed). Below it is showed an example:

epochs:
- 20
- 50
iwindow:(input timestamps)
- 20
- 50
learning_rate:
- 0.001
neurons:
- 80
owindow:(output timestamps)
- 10
- 20

The example above means that all possible combinations among the listed value will be evaluated and the one with the lowest validation loss will be chosen(for example one possible configuration can be 20 epochs, 20 input window, 0.001 of learning rate, 80 neurons and 10 output window and an other configuration could be 50 epochs, 50 input window, 0.001 of learning rate, 80 neurons and 10 output window; the one with the lowest validation loss will be chosen among all the possible candidates). *first_period* makes sense only if you choose *MultiModalAttention* like architecture. To

understand the meaning of this hyperparameter refer to the paper *Multi-Horizon Time Series Forecasting with Temporal Attention Learning* or the *algorithms'documentation* in the root folder. Consider that the implementation allows only a 2-modal attention mechanism. *starting_mask* makes sense only for variable horizon model; it is the starting value of the percentual of the output window that will be predicted. It is an integer between 0 and 1. The tip is to choice 0.2 when the output window is too long and 0.8 when the output window is short. The variable horizon model will start from this value to mask the output and will learn the best percentual of output window to predict. For example *starting_mask* equal to 0.1 and *owindow* equal to 10 means that only one timestep in the future will be predicted and the other 9 will be setted to 0 because considered not reliable. After the training the correct percentual of predictions will be estimated. Returning to the random search algorithm,in the case that a lot of different values will be settled for each hyperparameter the set of all possible configurations will result too high. To avoid long time computations it is possible to eliminate randomly subsets of configurations among all possible ones; This is the meaning of the *percentage_to_eliminate*. To allow the computation on all possible configurations the letter has to be set to 0.

Preprocessing

Figure A.1: When it is chosen a subset of columns the new column indexes are counted considering the new columns. The same is true when are obtained the forecasted columns.

Choose_columns allows to choose a subset of columns that become the input data of the model (the subset of variables of the multivariate time series). From this moment the index of columns is computed considering these chosen columns. With *input_columns* is possible to set the index of columns chosen if *choose_columns* is True. If *standardize* and *isallstd* are true all columns will be transformed into variables with 0 mean and 1 std. There is the possibility to not standardize all columns but only a subset of them using *columns_to_standardize* and setting *isallstd* to False. If *normalize* is true your data will be transformed into variables between 0 and 1 using the same procedure of *standardize*. Similarly it is possible to denoise variables with wavelets using exactly the same logic; the only difference is that is possible to choose

the mother wavelet using *wavelet_family*. To know all the possible wavelet's family use the link `https://scikit-image.org/docs/0.18.x/` Finally you can decide to smooth variables using B-splines, the procedure to use is very similar to that already explored. With *smoothness_preprocessing* you can regulate the amount of smoothness: 0 is interpolation, high number is a very smoothed curve. It is important to highlights that dataset are numpay arrays with timestamps on rows and variables on columns. Below it is reported an example where the columns 2, 3 and 4 are chosen like input from a dataset and (considering the new column index computed on these columns) the first column will be normalized and the third one will be denoised with wavelets (indexes start from 0):

choose_columns: true
input_columns:
- 2
- 3
- 4
standardize: false
columns_to_standardize:
- 1
isallstd: true
normalize: true
columns_to_normalize:
- 0
isallnorm: false
denoising: true
columns_to_denoise:
- 2
isall: false
wavelet_family: db1
smoothing: false
columns_to_smooth:
- 0
isallsmooth: false
smoothness_preprocessing: 20

Note that when *standardize* is False the columns to standardize and *isallstd* are ignored, but when the first option is set to True is important to choose *isallstd* to False if you want standardize only a subset of columns(the same it is true for normalizing, denoising...)

General Settings

With *forecasted_columns* you can choose output variables. Remember that the index to use is that one computed based on the chosen columns among all the columns. To continue the above example we choosed the columns 2,3,4 which new indexes are reset to 0,1,2; among these columns if you want to forecast the last 2 columns it should be taken the list of indexes: 1,2. Fill *path_train* and *path_test* with a path of a numpy array with timestamps on rows and variables on columns. With *stream* it is possible to attribute a name to the forecast. When *train* is true the model will be trained. If it is false it means that the model is already trained and should be only imported.

Plotting Input Data

*Variable_to_plot*is a list that defines the indexes of variables from the dataset of chosen columns to plot. Reusing the above example, we had columns 2,3,4. To plot the first of these you have to set *Variable_to_plot* to 0. (Remember always that the index is reset). *is_train* is a boolean, true means that will be plotted only data train, false means that will be plotted only data test. *input_wind_plot* is the length of each plot window in the subplot of input data. For example with a time series of 720 timestamps it could be possible to choose *input_wind_plot* equal to 350 and the plot results in two windows; the last 20 timestamps won't be plotted. It is important to highlight that it is not possible to plot all the timestamps (720 in this case) in only one window of a length of 720 because of a bug that should be correct in future. In "PlotInputs" folder visible in the root folder will be placed the plots.

Plotting Forecasting

forecasted_variable_to_plot is a list of indexes of forecasted variables to plot. Using the example above of two forecasted variables, to plot both of them, you should set:

forecasted_variable_to_plot:
- 0
- 1

In "Plotresults" folder visible in the root folder will be placed the plots.

A.3.2 Outlier Detection application

As told in the above paragraphs imagine having a dataset with different variables on the columns and with different measurements over time on the rows, by outlier detection we mean the possibility of finding anomalous patterns, not expected, of the variables along the time. The network in the training phase learns to recognize only the sequences without anomalies and in the testing phase it will be presented with data in which there may be anomalies and being patterns that the network has never seen, so it will be able to recognize them as outliers. One way to accomplish this analysis is Error Based Method which is used to forecast the behavior of variables on future timestamps and to make a comparison among forecasted values and true values; using algorithms that are inspired on papers that are referenced in the end of the documentation it is possible to label points as normal or anomalies. To know all the details of algorithms used you can check on the "algorithms' documentation" that you find in the root folder. As an alternative it is possible to use the Discriminator of GANs: the Generative Adversarial Networks (GANs) learn the distribution of true data and in particular the discriminator is able to decide if the data come from the true distribution or not (in this case we have anomalies). It is important to notice that before pass the time sequence to the GANs the spectrogram is computed using wavelets so a 2-dimensional input is passed to the model that contain time-frequency information. To use the Error Based Methods it is important to configure parameters like number of input timestamps, number of output timestamps, variables to get in inputs, variables to forecast and other hyperparameters that define the neural models. It is possible thanks to the "inputOutDet.yml" file that you find in the path "root\input_configuration". The following is the description of the settings of the "inputOutDet.yml". First of all, to use the Error Based Method it is important to set *OutlierDetection*(in "inputOutDet.yml") equal to True and verify that *OutlierDetectionGANs*(in "inputOutDet.yml") and *Forecasting* (in "inputForecast.yml") are settled to False. *architecture* can have input like: Encoder-Decoder, Attention, MultiModalAttention. To know the details of the algorithms you can check in *algorithms' documentation* placed in the root folder. The hyperparameter *layers* is an integer (not a list of integers, only one value is allowed!) and makes sense only for Encoder-Decoder. This is the number of encoder layers that is equal to the number of decoder layers. For the next settings of the architecture it is allowed to choose a list of values for each hyperparam-

eter. An algorithm called Random Search will find the best configuration (the one with the lowest validation loss) among all the possible configurations(cartesian product of all the listed values). Below it is showed an example:

epochs:
- 20
- 50
iwindow:(input timestamps)
- 20
- 50
learning_rate:
- 0.001
neurons:
- 80
owindow:(output timestamps)
- 10
- 20

The example above means that all possible combinations among the listed value will be evaluated and the one with the lowest validation loss will be chosen(for example one can be 20 epochs, 20 input window, 0.001 of learning rate, 80 neurons and 10 output window and an other configuration could be 50 epochs, 50 input window, 0.001 of learning rate, 80 neurons and 10 output window; the one with the lowest validation loss is chosen). *first_period* makes sense only if you choose *MultiModalAttention* like architecture. To understand the meaning of this hyperparameter refer to the paper *Multi-Horizon Time Series Forecasting with Temporal Attention Learning* or *algorithm' documentation* in the root folder. Consider that the implementation allows only a 2-modal attention mechanism. In the case that a lot of different values will be settled for each hyperparameter the set of all possible configurations will result too high. To avoid long time computations it is possible to eliminate randomly subsets of configurations among all possible ones; This is the meaninig of the *percentage_to_eliminate*. To allow the computation on all possible configuration the letter has to be set to 0. *z min* and *z max* are used to set the threshold to identify anomalies. Similar to other parameters to study the details of the methodologies refer to *algorithms' documentation*. As alternative to the Error Based Method there is the GANs model. To use it set *OutlierDetection* and *Forecasting* (in "inputForecast.yml") to False and *OutlierDetectionGANs* to True and *architecture* equal to GANs.

The hyperparameters useful for the Generative Networks are: *epochs*, *iwindow*, *latent_dim*, *n_filter_generator*, *n_filter_discriminator*. These are all integers (NOT list of parameters! The Random Search is not performed on the GANs!). Pay attention on the fact that *iwindow* has to be a multiple of 4 because of the implementation of the model. The generator and the discriminator have number of filters parameters because the core of the network is a convolutional neural network. The *latent_dim* is the dimension of the space that the generator maps in the space of the real distribution. As the Error Based Method the generative nets allow multivariate time series analysis; the interested variables are chosen in the preprocessing phase.

Preprocessing

Figure A.2: When it is chosen a subset of columns the new columns indexes are counted considering the new columns. The same is true when are obtained the forecasted columns.

Choose_columns allows to choose a subset of columns that become the input data of the model. From this moment the index of columns is computed considering these chosen columns. With *input_columns* is possible to set the indexes of columns that you want to select if *choose_columns* is True. If *standardize* and *isallstd* are true all columns will be transformed into variables with 0 mean and 1 std. There is the possibility to not standardize all columns but only a subset of them using *columns_to_standardize* and setting *isallstd* to False. If *normalize* is true your data will be transformed into variables between 0 and 1 using a similar logic as *standardize*. Similarly it is possible to denoise variables with wavelets using exactly the same logic; the only difference is that is possible to choose the mother wavelet using *wavelet_family*. To know all the possible wavelet's family use the link `https://scikit-image.org/docs/0.18.x/` Finally you can decide to smooth variables using B-splines, the procedure to use is alike the ones already explored. With *smoothness_preprocessing* you can regulate the amount of smoothness: 0 is interpolation, an high number is a very smoothed curve. Below it is reported an example where the columns 2, 3 and 4 are chosen like input and considering the new column index computed on these columns the first will be normalized and the third will be denoised with wavelets:

choose_columns: true

input_columns:
- 2
- 3
- 4
standardize: false
columns_to_standardize:
- 1
isallstd: true
normalize: true
columns_to_normalize:
- 0
isallnorm: false
denoising: true
columns_to_denoise:
- 2
isall: false
wavelet_family: db1
smoothing: false
columns_to_smooth:
- 0
isallsmooth: false
smoothness_preprocessing: 20

Note that when *standardize* is False the columns to standardize and *isallstd* are ignored, but when the first option is set to True is important to choose *isallstd* to False if you want standardize only a subset of columns(the same it is true for normalizing, denoising...)

General Settings

With *forecasted_columns* you can choose output variables. Remember that the index to use it that one computed based on the chosen columns among all the columns. To continue the above example we choosed the columns 2,3,4 which new indexes are reset to 0,1,2; among these to forecast the last 2 columns it should be taken the list of indexes: 1,2. Now the indexes of columns of the forecasted series are reset to 0,1. *outlier_detection_column* is the index (considering the dataset made of forecasted columns) that you want to monitor for outlier detection purpose. To continue the above example there are available 2 forecasted columns with reset column indexes 0 and

1; to perform outlier detection on the first variable *outlier_detection_column* should be equal to 0. In *path_test* you should insert the path of data on which analyze possible anomalies. Instead fill *path_train* with the path of train data with a numpy array *without* anomalies. The model learns normal sequences during train and the different behavior of anomalies in test data is labeled as anomaly. Remember that the inputs are numpy array with timesteps on rows and variables on columns. With *stream* it is possible to attribute a name to the variables monitored. *smooth* allows smoothing for the error among predicted and actual values. A value equal to 0 is not smoothing, an high value smooths a lot the error. When *train* is true the model will be trained. If it is false it means that the model is already trained and should be only imported.

Plotting Input Data

Variable_to_plot defines the indexes of variables from the dataset of chosen columns to plot. Reusing the above example, we had columns 2,3,4. To plot the first of these you have to set *Variable_to_plot* to 0. (Remember always that the index is reset). In order to plot training data *is_train* has to be equal to True, to plot test data it should be False. *input_wind_plot* is the length of each plot window in the subplot of input data. For example with a time series of 720 timestamps it could be possible to choose *input_wind_plot* equal to 350 and the plot results in two windows; the last 20 timestamps won't be plotted. It is important to highlight that it is not possible to plot all the timestamps (720 in this case) in only one window of a length of 720 because of a bug that should be correct in future. In "PlotInputs" folder visible in the root folder will be placed the plots.

Plotting Outlier Detection

wind_plot is the length of each plot window in the subplot. For example with a time series of 720 timestamps it could be possible to choose *input_wind_plot* equal to 350 and the plot result in two windows; the last 20 timestamps won't be plotted. It is important to highlight that it is not possible to plot all the timestamps (720 in this case) in only one window of a length of 720 because of a bug that should be correct in future. *monitored_variable_to_plot* is the variable to monitor in order to perform outlier detection. To plot this variable you should consider the indexes of *chosen columns*. Using the example above

where we chose columns: 2,3,4 with reset indexes: 0,1,2, to monitor the last one you should set:

monitored_variable_to_plot:

- 2

In "Plotresults" folder visible in the root folder will be placed the plots.

Bibliography

[1] Chenyou Fan et al. “Forecasting with Temporal Attention Learning”. In: *The 25th ACM SIGKDD Conference on Knowledge Discovery Data Mining (KDD’19)* (2019).

[2] Ian J. Goodfellow et al. “Generative Adversarial Nets”. In: *stat.ML* (2014).

[3] Ilya Sutskever et al. “Sequence to Sequence Learning with Neural Networks”. In: *Advances in neural information* (2014).

[4] Kyle Hundman et al. “Detecting Spacecraft Anomalies Using LSTMs and Nonparametric Dynamic Thresholding”. In: *The 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining* (2018).

[5] Sepp Hochreiter et al. “Long Short Term Memory”. In: *Neural Computation* (1997).

[6] Dzmitry Bahdanau. “Neural Machine Translation by Jointly Learning to Align and Translate”. In: *ICLR* (2015).

[7] Carl De Boor. “On Calculating with B-Splines”. In: *journal of approximation theory* (1970).