Deep Learning Methods for Weather Prediction

By

Ange-Clement AKAZAN (angeclement.akazan@aims.ac.rw)
African Institute for Mathematical Sciences (AIMS), Rwanda

Supervised by: Dr. D.Sc. Abebe Geletu W. Selassie

German research chair, Aims Rwanda

Co-supervised by: Dr. Kameni Nteutse Peguy

Postdoctoral scholar, Aims Rwanda

June 2022

AN ESSAY PRESENTED TO AIMS RWANDA IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF MASTER OF SCIENCE IN MATHEMATICAL SCIENCES

AIMS African Institute for Mathematical Sciences RWANDA

DECLARATION

This work was carried out at AIMS Rwanda in partial fulfillment of the requirements for a Master of Science Degree. I hereby declare that except where due acknowledgment is made, this work has never been presented wholly or in part for the award of a degree at AIMS Rwanda or any other University.

Student: Ange-Clement AKAZAN

ACKNOWLEDGEMENTS

Thank you for your favors and support in my life, my God.

I would like to express my deepest gratitude to my supervisor, Dr. D.Sc. Abebe Geletu W. Selassie, for his patience and support. I thank him for being there and available for me during this research. I also thank my co-supervisor, Dr. Kameni Nteutse Peguy, for his outstanding help and valuable advice in finishing this project. A special thank you is given to Dr. Cornelie Mitcha Malanda for her help and sound advice. I am very grateful to my entire family for their continued support and belief in me. In addition, I thank my wonderful colleagues at AIMS Rwanda; Without them, my time at AIMS Rwanda would have been extremely difficult. Finally, I would like to express my appreciation to AIMS Rwanda since it was an honor to be a student of this institute.

DEDICATION

This work is dedicated to my late father AKAZAN Kouassi Alexis, my mother Bosso Adjoua odile spouse AKAZAN, my old sister AKAZAN Jocelyne Euphrasie, my young sisters AKAZAN Reine Edith, AKAZAN Georgette Sophie and AKAZAN Christiane Joelle and my young brother AKAZAN Jean Hervé. I also dedicate this work to all those throughout the world who have been affected by weather disasters.

Abstract

The scientific field of anticipating atmospheric conditions at a given location and time is known as weather prediction. Predicting the weather is an important endeavor for scientists because it warns people about potential weather risks, allowing them to prevent weather-related losses. However, due to the chaotic structure of the atmosphere, existing model-driven weather forecast approaches struggle to accurately model the atmospheric conditions. Therefore, with the goal of providing an efficient tool capable of modeling atmospheric conditions and making credible predictions as quickly as feasible, this work focuses on the application of deep learning algorithms in weather prediction. In this thesis, we propose using two recurrent neural network algorithms, LSTM and Bi-LSTM, on a meteorological daily data set collected from Abidjan city in Ivory Coast, to predict three weather components: the temperature at 2 meters above sea level, wind speed at 2 meters above sea level, and precipitation for a one-day forecast using fourteen past observations. Afterward, We used three performance metrics to assess the prediction skills of both algorithms, and we discovered that both LSTM and Bi-LSTM have a mean absolute percentage error less than 18%, a mean squared error less than 0.35, and a mean absolute error less than 0.45, in temperature and wind speed prediction. These are quite good results produce by our algorithms nevertheless, they were unable to predict precipitation as good as the other variables. Based on the prediction visualizations, LSTM and Bi-LSTM appear to have difficulty in predicting the extreme values of the provided variables. Nonetheless, despite these shortcomings stated, the overall performance of the proposed recurrent neural network methods in weather prediction is found very promising.

Keywords: Weather, temperature, wind speed prediction, precipitation, deep learning, recurrent neural network, time-series data, LSTM and Bi-LSTM

Contents

List of Figures

List of Tables

1. Introduction

1.1 Motivation

Weather is the collection of phenomena occurring in our atmosphere on a daily basis. Different parts of the world and even a country may have different weather, which may change in minutes, hours, days, and weeks. Temperature, precipitation, wind, atmosphere pressure, humidity, and cloud formation are the 6 principal components of weather. Therefore, to talk about the weather at the specific location is also talking about each of these components at this location. The troposphere, as the lowest portion of the atmosphere, is where the vast majority of meteorological events occur. Weather forecasting is essential for human existence and economic development in every part of the world, particularly in Africa, because our continent is one of the most affected by weather extreme events due to a lack of scientific resources. Indeed, between 1970 and 2019, around 1.7 thousand natural catastrophes were documented in Africa (Kamer, 2022). In terms of human life, it has caused around 730 thousand deaths and droughts were by far the most common cause of mortality from natural catastrophes, accounting for 95% of all deaths. In the economics sector: weather disasters have caused around 10 billion USD in damages between 1980 and 2015 in southern Africa (Davis-Reddy and Mambo, 2017). Also, in the agriculture sector, farmers in Africa relying their ideas about the weather on their intuitions gained from experience in the agricultural field, they do not take into account the climate change effects. As a result, weather disasters such as drought, flood, and other natural disasters have a significant influence on agriculture in Africa, which lead to famine, disease, poverty, and so on.

As a result, it is worth noting that weather prediction is essential for the entire planet and more than necessary for Africa because it provides sufficient information about weather extreme events to reduce weather-related losses and enhance social benefits such as life and property protection, public health and safety, economic prosperity, and livability. To forecast the weather, several meteorological observation stations are used by researchers for collect meteorological data. This collected data is also coupled with cloud movement photos from satellites, providing a starting point for forecasting how the weather will evolve. Weather prediction is investigated using seven nonlinear partial differential equations(PDE) called Numerical Weather Prediction (NWP) equations, applied to the data obtained by observation.

NWP models represent fundamental physical processes in the atmosphere, and their impact on the temporal evolution of the atmosphere states. In order to produce the solution of prognostic equations in NWP models, geographical and temporal grids must be defined. All phenomena of the atmosphere (and at the surface) that are relevant for the evolution of future weather, might be fully described in a grid, by related model equations. A realistic simulation involves a description of processes that have a Spatio-temporal structure below the size of the model grid resolution. Several physical processes in the atmosphere such as cloud formation and the interaction of solar radiation with cloud droplets occur on very small spatial scales that NWP models cannot really address explicitly (Shutts and Palmer, 2007). The influence of these unresolved processes on model variables must be approximated using so-called parameterization approaches. Note that in climate modeling, the parameterization method is a means of employing a simplified process to represent another process that is too small-scaled or complicated to be physically modeled.

The atmosphere is vast and complicated and due to the chaotic nature of its states, modeling the atmosphere is challenging. Therefore, mathematical models based on physical laws are sometimes not flexible enough to model randomness in the atmosphere behavior, which leads to errors in the model result. When the original forecasting equations are derived from their full theoretical form and converted to computer code, the computer translations of that model forecast equations are not able to capture all of the features at all resolutions. This conversion process leads to errors in the computer model result called modeling errors (Bochenek and Ustrnul, 2022).

As a result, even if perfect measurements were available and the initial condition of the atmosphere was known precisely, some information about atmospheric fields will be omitted or misinterpreted by NWP models (Ren et al., 2021). Also, the use of a numerical approach to solve a nonlinear differential equation is computationally expensive. Indeed, to be implemented on the large surface scale, NWP models require the use of supercomputers. A specific subset of machine learning was targeted to reduce the numerous errors produced in weather prediction and make weather prediction more accurate and reliable. This subset is a powerful data modeling tool that can capture and express complicated interactions between inputs and outputs. It is essentially a system of interconnected networks that work together to effectively identify, categorize, and characterize elements in the data of interest. This subset includes an error-correcting mechanism, which is required to master the unpredictability of the atmospheric state and reduce the number of possible mistakes and deep learning is the name given to this subcategory. Because of their ability to learn time series data, we recommend employing recurrent network techniques and specifically LSTM and Bi-LSTM for predicting weather components as well as possible.

1.2 Problem statement

Atmospheric conditions fluctuate continually throughout time and can be rather highly variable at times. As a result, NWP models may exhibit an insufficient understanding of physical causes and difficulties in extracting usable knowledge from a flood of observation data. Also, for small surface areas, due to the lack of surface weather data and the limitation of computational resources, weather forecasting becomes hard to investigate with NWP models. Furthermore, NWP models are computationally costly and need large computer resources, resulting in a long calculation period before producing outcomes and limiting the weather forecasting industry to only strong climate research groups with powerful computing resources. Therefore, regardless of the data obtains, any individual can not fully utilize NWP to forecast the weather in a specific region.

1.3 Objective

1.3.1 Main objectives. The purpose of this essay is to demonstrate the value of deep learning algorithms in localized weather prediction. We intend to show that deep learning algorithms may be a helpful and accessible tool for forecasting weather.

1.3.2 Specific objectives.

- To study Long-Short Term Memory (LTSM) neural network and Bidirectional-LSTM (Bi-LSTM) for weather prediction
- To use and perform Long-Short Term Memory neural network over several weather components and compare its quality results in function of the weather component selected.
- To analyze and evaluate the performance of recurrent network algorithms in weather prediction

1.4 Contribution

Our major contribution to this study is to use two recurrent neural network algorithms for predicting three weather components, as well as testing the prediction skills of those approaches and examining the forecast quality by comparing graphically the predicted values to the real observations. In this thesis, we utilized a meteorological data from Abidjan, Ivory Coast, that included 2557 observations and 7 variables, and we used LSTM and Bi-LSTM algorithms to predict those variables for one day using 14 days of historical observations. The variables selected are: Temperature at two meters above sea level, wind speed at two meters above sea level, and precipitation. Following that, the prediction abilities of both algorithms were assessed and based on the results of three performance metrics, both algorithms were found to be quite accurate in forecasting temperature and wind speed but they performed badly in predicting variable precipitation.

1.5 Outline

The rest of this thesis is structured as follows: **Chapter 2**, will introduce the reader to the background, where we define some important notions for our research and followed by the literature review. **Chapter 3** describes mainly the deep learning algorithms used (LSTM and Bi-LSTM) starting with the explanation of the concept of recurrent neural network, passing by LSTM concept, and ending by Bi-LSTM description and the last section will be reserved for the performance metrics used to assess our models. **Chapter 4** provides a description of the data set, the implementation step, an implementation flow chart, and the findings of both deep learning approaches (LSTM and Bi-LSTM techniques) in temperature, wind speed, and precipitation prediction. This chapter also performs a comparative study to assess their weather predicting abilities using the collected data as a baseline. **Chapter 5**, will synthesize the results and provide a comprehensive conclusion, and state future research direction.

2. State-of-the-art

The background and literature review are the emphasis of this chapter. To be more explicit, the notion of a model-based approach in climate is provided in the background, and deep learning is briefly outlined. Following that, we addressed artificial neural networks, which serve as the foundation for deep learning, and then we described what time-series data is. The last section is a literature review in which we discuss the history of the weather forecasting problem in climate science as well as several weather prediction methodologies.

2.1 Background

2.1.1 Model-based approach. In climate modeling, the prediction is governed by mathematical models. These models are comprised of several PDEs that represent the interaction between atmospheric components.

2.1.2 Deep Learning. Deep learning is a subset of machine learning, that try to learn feature like the human brain by using neural network algorithms that enable computer algorithms to learn and improve on their own.

2.1.3 Artificial neural network. Neural networks are built up using several layers and each contains numerous neurons or nodes, similar to how the human brain is made up of neurons. Individual layer nodes are linked to nodes in neighboring layers. The number of layers in the network indicates how deep it is. In the neural network, the information is transmitted from layer to layer and each information transmission is assigned to weight. The nodes with higher weights will have greater influence in the layer. The last layer assembles the weighted inputs to generate an output. The artificial neural network process is very well represented in fig 2.1

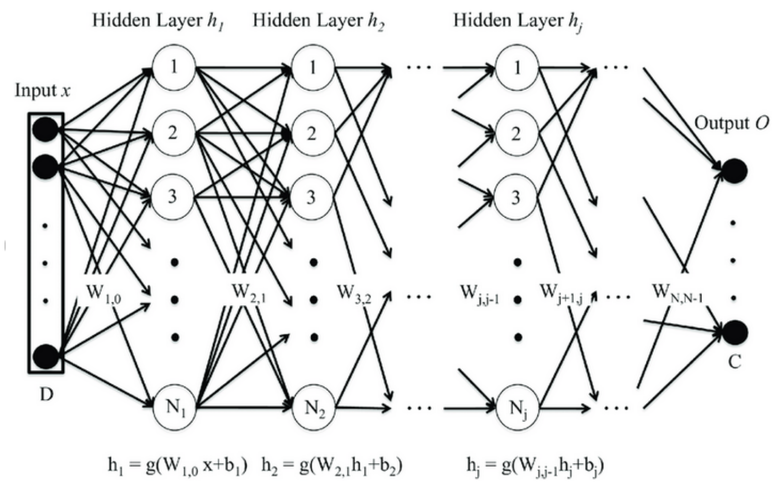

Figure 2.1: Deep neural network architecture Lozano-Diez et al. (2017)

2.1.4 Time series data. A time series is a data set that demonstrates how a variable or a group of variables vary over time. It serves as a foundation for forecasts in climate science by demonstrating how the weather fluctuates throughout time.

2.2 Literature Review

There is a panoply of work done in the area of weather predictions. In this part, we will talk about some previous works done about the essay topics, relating the history of weather prediction from the birth of NWP models to the application of machine learning in weather prediction. The concept of weather prediction is mainly dominated by the use of the NWP models because of the seniority. Indeed, Willis and Hooke (2006) affirmed that the starting point of the climate modeling is this sentence "meteorology is essentially the application of hydrodynamics and thermodynamics to the atmosphere" from the American meteorologist Cleveland Abbe in 1890. Bjerknes et al. (2009) explains that in 1904 Vilhelm Bjerknes gave a formal scientific approach to the weather forecast. He affirmed that to use physical laws for weather prediction we must be able to determine the atmosphere state at a given time (initial states) and master the laws that govern the transition of the atmosphere states. But, at this time, he knew that the initial state of the atmosphere could not be obtained because of the lack of weather measurement data. Nonetheless, he stated that if temperature, pressure, velocity, humidity, and density could be obtained at any time, the atmosphere's state could be completely modeled. He, therefore, proposed seven differential equations (primitive equations) to model the atmosphere's state using physical laws such as mass conservation and thermodynamic laws, among others.

In 1915, Lewis Fry Richardson used a finite difference method to solve numerically the Bjerknes's equations. Because of the lack of data at that time, the author used an arithmetic method to get some data for the implementation of his methods, but unfortunately, he obtained unrealistic results. However, he found that the initialization of the atmosphere state was an important issue and that it was not possible to integrate this in the IAS computer. Afterward, in order to simplify the primitive equations Charney et al. (1950) proposed the vorticity equations, using a scale analysis, and the result was called "quasi-geostrophic system" but solving it analytically required too many assumptions over the parameters of the equation. Therefore, Charney et al. (1950) chose to solve these equations numerically using the finite difference method. The computer code of this method was implemented into the most powerful computer at this time: the Electronic Numerical Integrator and Computer (ENIAC). Fortunately, the result of four forecasts of 24 hours via ENIAC was satisfying. Considering, that the climate change was a global issue, a family of models called the General circulation model (GCM) was introduced. The first GCM was proposed by Phillips (1956). He did so with a two-level quasi-geostrophic model that starts with a relatively steady atmosphere. At times, this model has shown to be effective, such as when predicting the easterly-westerly-easterly distribution of a surface zonal wind. However, several of the characteristics were unrealistic in terms of physical principles. He has affirmed that the failure in his model is due to the simplicity of the latter. Another GCM was developed at the National Center for Atmospheric Research (NCAR) by KASAHARA and WASHINGTON (1967). In this model, the height was used as a vertical coordinate for the first time in atmosphere modeling. This model was one of the first to include physical processes such as short wave radiation, longwave radiation, and so on. However, like with many other models, one of the key issues with this model was its inefficiency in investigating small surface areas. This marks the arrival of Regional Climate Models (RCM) in climate modeling. Kida et al. (1991) created an RCM by nesting a high-resolution Limited Area Model (LAM) inside a low-resolution GCM. The model result was pretty good. But this model relies on the collaboration between LAM and GCM however during a long-term integration these two models tend to be different. Note that the meteorologists Syukuro Manabe and Hasselmann were named co-laureates of the Nobel Prize in Physics in 2021 for their studies on the effects of carbon dioxide on world temperature, which led to credible climate change forecasts. Despite their utility in weather prediction, NWP models faced a number of difficulties. Some major challenges for NWP models,according to Willis and Hooke (2006), include parameterization, consistent initial conditions determination, and the complexity of the data assimilation approach. However, he did not offer a consistent answer to these problems. Some scientists tried to fix certain issues of NWP models. Pielke Sr et al. (2006) for example, aimed to solve some of the issues with NWP models. He stated that appropriately parameterizing physical processes in NWP models requires high computational cost, and he proposed a computationally efficient approach for accurately parameterizing physical processes in NWP models. It should be noted that the solutions provided by Willis and Hooke (2006) are, in general, quite difficult to implement. As a result, rather than relying solely on NWP models, scientists developed a statistical approach known as statistical post-processing to improve NWP results. In that vision Gneiting et al. (2005) suggested using the Ensemble Model Output Statistics (EMOS) to correct forecasting bias in NWP prediction results. This technique was based on a multiple linear regression estimation. The EMOS approach was used to forecast pressure at sea level and surface temperature for 48 hours, and approximately the general measurements resulted in a root-mean-square error of 9% and a mean absolute error of 7%, which were better than the initial NWP models used in his study. But this model was a model distribution-based assumption. Bremnes (2004) proposed the use of quantile estimation for making reliable precipitation predictions. He firstly estimated the probability of precipitation and secondly, he estimated certain quantiles in the distribution of precipitation quantities based on precipitation occurrence. The result was not very good compared to the NWP used. In fact, his method was not really complete since the quantile estimation was restricted to only one estimator. Not that several other calibration methods using statistical post-processing have been created and improved a little bit the prediction of NWP models.

In addition, to introduce statistical analysis to improve NWP models in weather prediction, a formal statistical approach has been used. Which is to use a statistical model to directly predict the weather. Since 1950 scientists had already started to use statistical methods for weather prediction. Note that, one of the first statistical methods used in weather prediction was linear regression. Klein et al. (1959) have derived a Linear multiple-regression model expressing 5-day mean surface temperature as a function of a 5-day mean 700-mb heights centered 2 days earlier, using a statistical screening technique. The 3-day forecast gave some good results but when it comes to 5-day forecast result the quality of the result was bad. This problem was due to the inappropriate use of linear regression. Also, Billet et al. (1997) utilized multiple regression methods and logistic regression to predict maximum hail size. Unluckily, the prediction was unsuccessful as expected; probably because of the wrong choice of the variables. Since the study of the weather and the climate required time-series data, several time series methods have also been used for weather prediction. As an example, let's cite Campbell and Diebold (2005) who used a simple time series method called GARCH methods, for predicting daily temperature at several locations. he has shown that time series methods are efficient to model weather derivatives and predict them over time. however, due to wrong management of the noise in the data the forecast result was not better than NWP models produced by EarthSat from a one-day to an eight-day forecast. In addition to the statistical methods, machine learning was also an alternative to the scientists for weather forecasting. Indeed, machine methods have been found efficient in the prediction field, therefore, it could be an interesting way to use machine learning in weather prediction. Bochenek and Ustrnul (2022) have presented an interesting review of work done in machine learning for weather prediction where they have concluded that machine learning is a significant tool for weather prediction. Deep learning methods also have been used in weather prediction, we take as an example Weyn et al. (2020) who used a deep convolution neural network (CNN) to predict some weather component on a global grid. This model gave a better result than a certain NWP model used in his study, for short and medium-range prediction. However, this deep CNN was outperformed by another NWP model that uses model calibration. The issue with the CNN approach is that there was insufficient data to train deep CNN models using a short-term forecast, as CNN requires a large amount of data to function successfully. Another paper worth mentioning is that of Hewage et al. (2019), who provided an LSTM model that outperformed an NWP (from the Weather Research and Forecasting organization) for short-term weather forecasting and surface weather data. To enhance the model, the authors advised increasing the number of parameters to estimate in the model. However, his proposition will result in a large increase in algorithm complexity. Also, Ponnoprat (2021) developed a specific sort of autoencoder model that can address seasonality and nonlinearity in precipitation forecasting, however, the model required a large amount of data for training in order to perform effectively. It is worth noting that several different deep learning approaches have been used in weather prediction, with some promising outcomes thus far, in this paper we will focus on LSTM and Bi-LSTM for temperature, wind speed, and precipitation forecasting in Abidjan (Ivory Coast).

3. Methodology

In this study, LSTM and Bi-LSTM will be employed for weather prediction. As a result, this chapter will introduce the reader to the LSTM and Bi-LSTM approaches. The first section provides the recurrent neural network concept as a background for understanding the concept of LSTM and Bi-LSTM. In the second section, the Long Short-Term Memory approach, its training procedure, and some of its modified forms will be described. Following that, in the third section we discussed the Bi-LSTM approach, and the final section is dedicated to the performance metrics.

3.1 Recurrent Neural Network

Recurrent neural network algorithms (RNN) are special types of artificial neural networks (ANN) that mainly work with time-series data or any kind of sequential data set. They are generally used to deal with temporal dependency problems in a data set and their distinctive characteristic is their special ability to learn sequential data using a memory system to gather prior knowledge from previous input and output. This prior knowledge is then utilized to influence the current input and output. RNN structure is comprised of the input layer, one hidden layer (Vanilla RNN) or more than one (Deep RNN), and the output layer. We shall concentrate on "Vanilla RNN" in this study because it is the foundation of all RNN modified versions.

3.1.1 Recurrent Neural Network Description. In general, for a neural network having only one hidden layer, when data is entered in the input layer and transmitted to the hidden layer, this hidden layer transfers its information to the output layer to get the model output. But, in RNN the hidden layer does an additional work which is to generate, at the current time t, a recurrent memory as a vector h_t called hidden state, used to enhance the prediction at the next time step. The layer of a RNN is called recurrent layer because it transfers information learned during the previous input analysis to the next input analysis to enhance its output. A neuron in the recurrent layer is known as a recurrent unit.

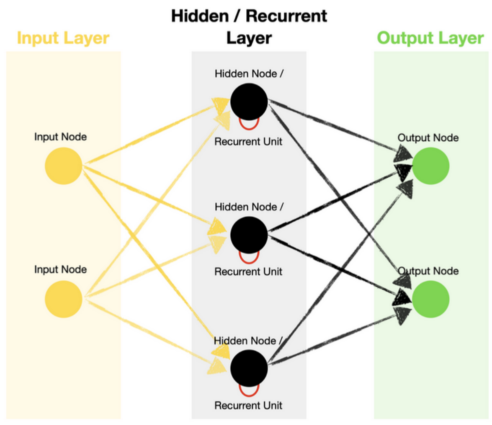

Figure 3.1: Standard RNN architecture (Dobilas, 2022)

3.1.2 RNN principle. Considering the time series data $X = \{x_1, \cdots, x_N\}$, $x_t \in \mathbb{R}^d$, where d represent the number of feature and N the number of time step $t \in \{1, ..., N\}$. RNN processes the data by ascendant time step, this means that it processes the data X starting from x_1 and ending by x_N. At each time step t, the algorithm generates a vector called hidden state h_t $\in \mathbb{R}^m$ and an output $\hat{y}_t$ that is either a vector or a scalar, depends on the problem studied. Note that the hidden state represents the information about the input data at time t (x_t). At each time step t, RNN transfers the information contained by h_t to itself at the next time step $t+1$. Once at time step $t+1$, RNN creates the hidden state h_{t+1} from h_t and x_{t+1} and therefore generates the output $\hat{y}_{t+1}$ using the current hidden state h_{t+1}. This process is repeated until the last input data x_N is processed The representation of RNN unfolded in fig 3.2 describes the RNN recurrent process.

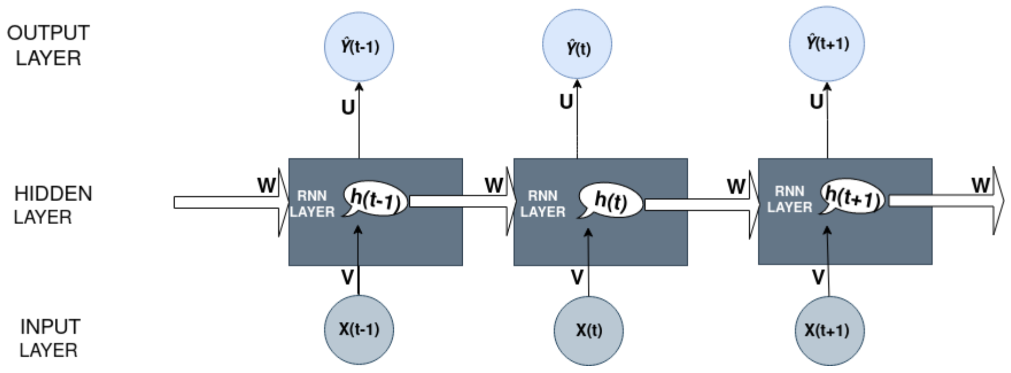

Figure 3.2: Unfolded RNN where U, V, W are weight matrices and $h(t)$ is the hidden state at current time t

3.1.3 RNN algorithm. In this part, we will use the notation of figure 3.2 by assuming that $h_t = h(t)$.

Algorithm 1 RNN Algorithm (Yin et al., 2017)

```
1:  $Data = \{x_1, \cdots, x_N\}$
2:  procedure RNN($h_0, Data$)                                  ▷ $h_0 \in \mathbb{R}^m$
3:      System Initialization
4:      $x_1 = Data[:, 1]$
5:      $h_1 = f_h(Vx_1 + Wh_0 + b_h)$
6:      System Iteration
7:      for ($t = 2;\ t \leq N;\ t{+}{+}$) do
8:          $x_t = Data[:, t]$
9:          $h_t = f_h(Vx_t + Wh_{t-1} + b_h)$            ▷ $b_y \in \mathbb{R}^l$, $h_t$ and $b_h \in \mathbb{R}^m$
10:         $\hat{y}_t = f_y(Uh_t + b_y)$
11:
        ▷ $U \in \mathbb{R}^{dim(x_t) \times dim(h_t)}$, $V \in \mathbb{R}^{dim(h_t) \times dim(x_t)}$ and $W \in \mathbb{R}^{dim(h_t) \times dim(h_t)}$
```

Note that at time-step t=0, the hidden vector h_0 is mostly considered as the null vector.

At time step t=1

The RNN receives the input x_1 and generates from the hidden layer, the hidden state h_1 using h_0 through the following formula

$$h_1 = f_h(Vx_1 + Wh_0 + b_h) \tag{3.1.1}$$

and the output of the network is

$$\hat{y}_1 = f_y(Uh_1 + b_y) \tag{3.1.2}$$

At time step t=2

The RNN receives as input x_2 and generates the h_2 from h_1 by the formula

$$h_2 = f_h(Vx_2 + Wh_1 + b_h) \tag{3.1.3}$$

and the output is

$$\hat{y}_2 = f_y(Uh_2 + b_y) \tag{3.1.4}$$

For time step t$\in \{2, \cdots, N\}$

When, the RNN receives the input x_t, it updates the past hidden state h_{t-1} in h_t using the formula

$$h_t = f_h(Vx_t + Wh_{t-1} + b_h) \tag{3.1.5}$$

This updated hidden state h_t is used to determine the output $\hat{y}_t$ through the relationship

$$\hat{y}_t = f_y(Uh_t + b_y). \tag{3.1.6}$$

Note that V is the weight matrix from the input layer to the hidden layer, W is the weight matrix connecting the h_{t-1} to h_t and U is the matrix connecting the hidden layer to the output layer. b_h and b_y are bias vector, f_h and f_y are both activation function (which vary according to the problem we are solving). The algorithm 1 is a recap of what has been written above.

3.1.4 Recurrent Neural Network Training. To train a recurrent neural network neural model we use gradient-based algorithms (see algorithm 2). But since each output of the RNN is time-related, scientists have introduced the notion of time to compute the gradient of the loss function considered. This method is known as Back-propagation through time. Considering a RNN under its unfolded form shown in fig 3.2, let $L_t(y_t, \hat{y}_t)$ be the loss function of the RNN at time step t (where $\hat{y}_t$ is the RNN predicted value and y_t is the real value). The total loss function of the RNN algorithm is defined as

$$L = \sum_t L_t(y_t, \hat{y}_t). \tag{3.1.7}$$

We use Backpropagation to compute the gradient of the total loss function with respect to the weight matrices and the biases (V, U, W, b_h, b_y). Therefore, as an example, let compute it for $\dfrac{\partial L}{\partial W}$ the Gradient of L with respect to W. From eq.(3.1.7), we deduce

$$\frac{\partial L}{\partial W} = \sum_t^N \frac{\partial L_t}{\partial W} \tag{3.1.8}$$

Using a chain rule we get with L_t we get $\forall\ t \in \{1, \cdots, N\}$,

$$\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\frac{\partial h_t}{\partial W} \tag{3.1.9}$$

Because h_t does not depend directly on W but depends on h_{t-1}, therefore, eq.(3.1.9) becomes

$$\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial h_{t-2}}\cdots\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{W} \tag{3.1.10}$$

Note that

$$\frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial h_{t-2}}\cdots\frac{\partial h_2}{\partial h_1} = \prod_{i=2}^{t}\frac{\partial h_i}{\partial h_{i-1}} \tag{3.1.11}$$

So eq.(3.1.10) becomes

$$\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\left(\prod_{i=2}^{t}\frac{\partial h_i}{\partial h_{i-1}}\right)\frac{\partial h_1}{\partial W} \tag{3.1.12}$$

Hence, putting eq.(3.1.12) in eq.(3.1.7) we get

$$\frac{\partial L}{\partial W} = \sum_{t}^{N}\frac{\partial L_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\left(\prod_{i=2}^{t}\frac{\partial h_i}{\partial h_{i-1}}\right)\frac{\partial h_1}{\partial W} \tag{3.1.13}$$

We then use the algorithm 2 to update W. The process is repeated for each weight matrix and bias vector of the RNN to accomplish the training.

Algorithm 2 Training Algorithm (Ruder, 2016)

```
1: System Initialization
2: set $k = 0$, fix an initial iterate $w^0$
3: set $d_0(w^0)$ and $\eta_0$
4: System Iteration
5: while $\|d_k(w^k)\| > \epsilon$ do
6:     $w^{k+1} = w^k + \eta_k d_k(w^k)$
7:     $k = k + 1$
8:     Update the step-length $\eta_k$ and evaluate $d_k(w^k)$
9:     end While
```

It is worth noting that when $d_k = -\nabla L(w^k)$, the algorithm 2 is referred to as **gradient descent**. It should be noted that L must be a differentiable function. There exists several algorithms for training RNN, and some of them are more sophisticated than gradient descent, some of which are more advanced than gradient descent, such as: Adam (Adaptive Moment Estimation), SGD (Stochastic gradient Descent), RMSProp (Root Mean Squared Propagation), AdaDelta (Adaptive Delta), AdaGrad (Adaptive Gradient Algorithm) and so on.

3.1.5 Limitations of RNN and introduction to LSTM. The idea behind the RNN process is that since the data is sequential, the information at the current time step t may depend on that of the previous time step $t-1$. Therefore, to generate the output of an input at any time-step t, we must generate a memory h_t taking into account the previous information x_{t-1} and h_{t-1}, and use it (memory h_t) to produce the current output $\hat{y}_t$. This memory generated using the recent lag version is called **Short-Term Memory** which is able to use information from the previous time step $t-1$ to predict output at time-step t. But, the problem with RNN is that it can not handle the problem of long term dependency between sequential data because of specific problems called vanishing gradient (the gradient used to update the weights is nearly equal to zero therefore no training is carried out since the weights are not updating) and exploding gradient (gradient increases exponentially, giving very large values to weights that are not helpful for training). This is why has been introduced the **Long Short-Term Memory** algorithm to generate a system called a gating system that discards useless information and keeps the useful ones to handle the problem that RNN was facing.

3.2 Long Short-Term Memory Based Weather Prediction

3.2.1 Problem description. A group of component of the weather (temperature, wind speed, specific humidity, and so on) is observed over time (at equal time intervals). These observations, combined, constitute a multivariate time series data $\mathcal{D}$. We restructure $\mathcal{D}$ using T observations of each variable as input and the next day value of specific variables as output. We therefore get a sequence of observational data $X_t = \{x_{t-T}, \ldots, x_{t-1}\}$, with $(x_{t-h})_{h=\{1,\ldots,T\}} = (x^1_{t-h}, \ldots, x^D_{t-h})^T \in \mathbb{R}^D$, where t is a timestamp, D represents the number of observed variables and T is the number of time step used to slide the data into observational and output data. Note that in our implementation T=14 days and D=7. Let y_t be the variable of future state of the variable x^d_t, $d = \{1, \ldots, O\}$. As weather is changing randomly, y_t may be seen as a stochastic process according to eq.(3.2.1)

$$y_t = x^d_t = f(x_{t-T}, \ldots, x_{t-1}, \varepsilon_t, \theta_t) \tag{3.2.1}$$

where $\varepsilon_t \sim \mathcal{N}(0, \sigma^2)$ is the noise of X, θ_t is a set of unknown parameter (Weights and biases). The function f is at prior unknown, in our analysis, we will use LSTM architectures to construct a good estimator $\hat{y}_t = \hat{f}(x_{t-T}, \ldots, x_{t-1}, \varepsilon_t, \theta_t)$ of the function f so that reliable values of y_t will be determined much more easily.

3.2.2 Long Short-Term Memory. Long Short Term Memory networks (LSTM) are special types of RNN algorithms created by (Hochreiter and Schmidhuber, 1997) that are able to learn long-term dependencies between sequential data, using a system of memory storage and information distribution called gating process, into each neuron of the LSTM neural network. These gates either open or closed allow information to be saved, written, updated, or forgotten in LSTMs and hold information in the 0-1 format according to their relevance.

3.2.3 Long Short Term Memory (LSTM) element description.

- **Gates:** Control the flow of information entering and leaving the memory. Each gate is a unit layer associated with the sigmoid function. They are controlled by concatenation of previous time step hidden state, actual input. Note that the sigmoid layer outputs are numbers between 0 and 1. An output close to zero means "This information does not pass through the gate" and a value close to 1 means "This information is important to pass through the gate"

- There are 3 principal types of gates in LSTM: Input gate, output gate, and forget gate
  1. **Forget gates**: Controls which information from memory can be discarded or validated.
  2. **Input gates**: The input gate is used to measure the significance of new information given by the input x_t
  3. **Output gates**: The output gate decides which output the system will display from the memory
- **Candidate cell state**: Contains the information candidate to be inputted in the memory by the input gate.
- **Cell state or Memory cell**: is the key of LSTM. It is the memory of the LSTM process. It undergoes several changes via the forget and input gate.
- The inputs of the LSTM hidden layer at all time step are the vectors: x_t(Current input), C_{t-1} (previous memory state) and h_{t-1} (hidden state at time $t-1$).
- The output of the LSTM process at all time step are the vectors: C_t (current memory state) and h_t (current hidden state) and $\hat{y}_t$ (output of the neural network at time step t)

3.2.4 unfolded LSTM version . The recurrent structure or unfolded version of LSTM is represented in the figure 3.3

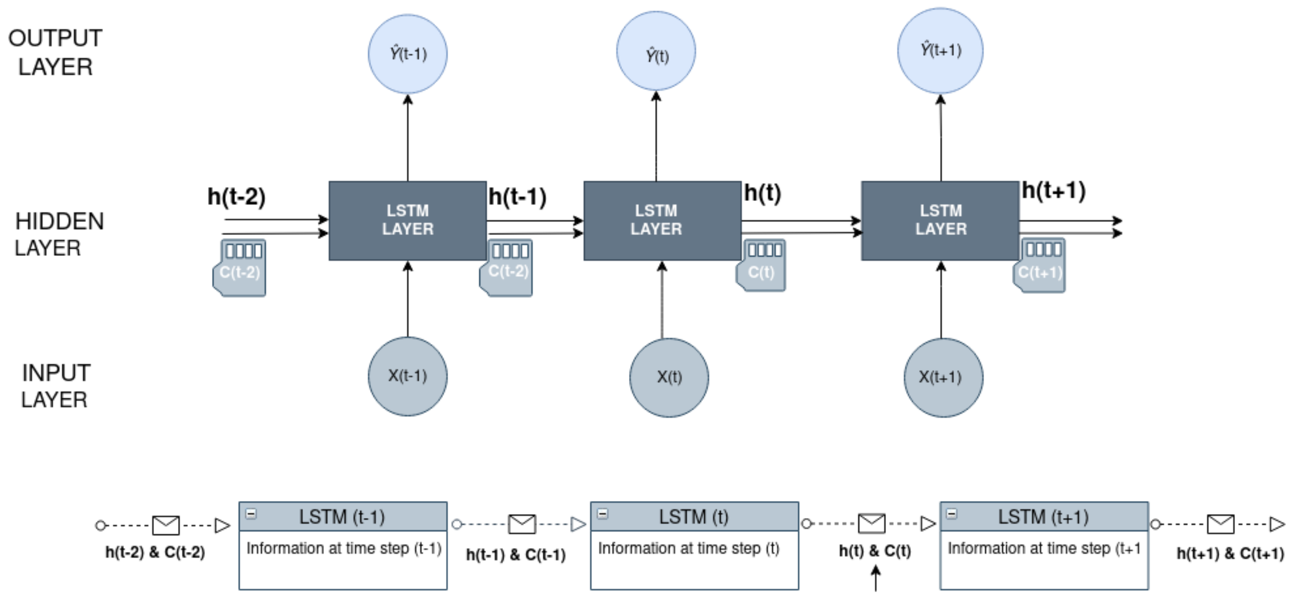

Figure 3.3: Unfolded LSTM where $C(t)$, $h(t)$ and $\hat{y}(t)$ represent respectively the memory cell state, the hidden state and the output of the LSTM model at time step t

3.2.5 Long Short-Term Memory (LSTM) Principle. The core of LSTM and the innovation that differs LSTM to RNN is The cell state or memory cell or again the long term memory. At time step t, It is defined by the formula

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{3.2.2}$$

where f_t, i_t are respectively the forget and input gates and $\tilde{C}_t$ is the candidate cell state. Note that f_t, i_t and $\tilde{C}_t$ are created to participate to the investigation of C_t. Note that the remaining part of this section described the step of the LSTM mechanism.

1. **First step**

   When the LSTM network receives an input x_t, firstly this input is concatenated with the previous hidden state h_{t-1} by creating the following vector

$$I_f = W_f^1.h_{t-1} + W_f^2.x_t + b_f \tag{3.2.3}$$

   (W_f^1 and W_f^2 are weights matrices and b_f is the bias vector). Afterwards, I_f is inputted in a layer called **Forget gate** containing a sigmoid activation function (σ). Note that the sigmoid activation is a vector function defined as follows

$$\sigma : \mathbb{R}^n \longrightarrow \mathbb{R}^n$$
$$(x_1, \cdots, x_n) \longmapsto \left(\frac{1}{1+\exp(-x_1)}, \dots, \frac{1}{1+\exp(-x_n)}\right).$$

   The forget gate at current time step f_t determines whether the information carried by the vector I_f will allow us to discard or keep information coming from the prior memory cell. This choice is done by using the formula:

$$f_t = \sigma(W_f^1.h_{t-1} + W_f^2.x_t + b_f).$$

   f_t outputs a vector that has the same dimension than the bias vector b_f, whose components are between 0 and 1. It should be noted that a component of f_t equal or close to 0 indicates that the information it provides is irrelevant and might be forgotten, but a value equal or close to 1 indicates that the information is significant so it is conserved by the gate, and may be utilized in the following steps. The representation of the forget gate is shown in figure 3.4

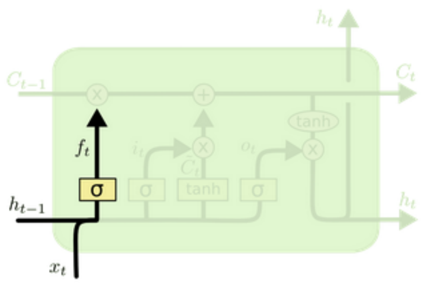

Figure 3.4: Forget gate description (Olah, 2015)

2. **Second Step**

   After the activation of the forget gate, another gate called **input gate** is triggered, taking the same parameter as the forget gate: h_{t-1} and x_t. This gate is used to select relevant information from a base of information candidates and store them in the current cell state or current memory C_t.

   - At each time step the input gate is defined as

   $$i_t = \sigma(W_i^1.h_{t-1} + W_i^2.x_t + b_i) \tag{3.2.4}$$

     where σ is the sigmoid function, W_i^1 and W_i^2 are weights matrices from the input gate layer and b_i is the bias vector assigned to it.

   - Just after the activation of the input gate, a layer called **candidate cell** $\tilde{C}_t$, using the same parameter with all gates, is activated. This layer uses $tanh$ activation function (gives a vector containing value between -1 and 1) as activation function. $\tilde{C}_t$ creates a vector of candidate information that could update the cell state (memory) C_t. The candidate cell is defined as following

   $$\tilde{C}_t = tanh(W_C^1.h_{t-1} + W_C^2.x_t + b_C) \tag{3.2.5}$$

     where W_C^1 and W_C^2 are weights matrices for the candidate gate layer and b_C is the bias vector assigned to that gate. Figure 3.5 gives the description of the input gate and candidate cell processes.

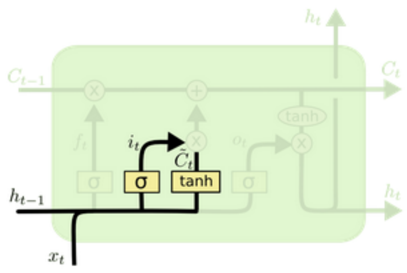

Figure 3.5: Input and candidate gates (Olah, 2015)

   - Having the input and output gates and the candidate cell state $\tilde{C}_t$, the forget gate filters the values of the previous cell state(memory) C_{t-1}, discards the irrelevant, and keep the important values into the current memory cell state C_t. Simultaneously, the input gate updates the candidate cell. These operations are described in the following formula

   $$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{3.2.6}$$

     Where * is the element-wise Multiplication (Hadamard product). Figure 3.6 display the candidate gate process.

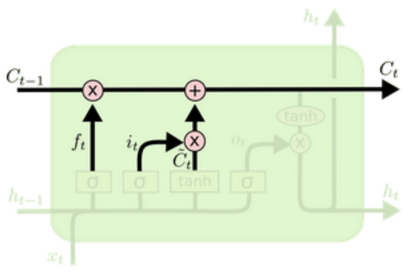

Figure 3.6: Cell state or memory construction (Olah, 2015)

3. **Third Step**

   In this step, the **output gate** is created. Taking as parameters h_{t-1} and x_t, the output gate decides to what information to output from the memory. The output gate is generated through eq.(3.2.7)

   $$o_t = \sigma(W_o^1.h_{t-1} + W_o^2.x_t + b_o) \tag{3.2.7}$$

   where W_o^1 and W_o^2 are weights matrices for the output gate layer and b_f is the bias vector assigned to that gate. This output gate is used to filter the information given by the current cell state C_t (current memory) when the validated information is created. This validated information represents the final information generated by LSTM at the current time t which is also called hidden state h_t. At each time step the current hidden state is

   $$h_t = o_t * tanh(C_t) \tag{3.2.8}$$

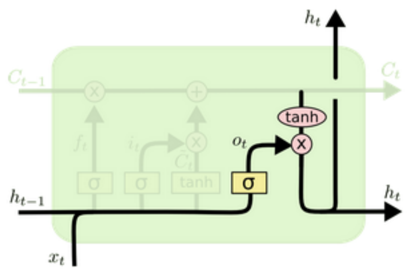

Figure 3.7: current hidden state and output (Olah, 2015)

   Note that generally, the final output of an LSTM network is h_t

4. **Fourth Step (Optional)** This step is optional. Rather than using h_t as LSTM's output, we may determine the output ($\hat{y}_t$) using h_t via the following formula.

   $$\hat{y}_t = f_y(W_y h_t + b_y) \tag{3.2.9}$$

   Where f_y is an activation function, W_y is a weight matrix for the output gate layer and b_y is the bias vector assigned to that gate

3.3 LSTM Training

To train an LSTM algorithm, we use a gradient based-algorithm via Back-propagation through time. Note that the training of LSTM is based on that of RNN that we have explained in section 1 of this chapter. The only difference between them is that LSTM has much more weight matrices and biases vector to updates because of the additional layer that it considers for its implementation. Indeed, the weight matrices and bias vectors in LSTM to update are from the forget gate (W_f^1, W_f^2, b_f), input gate (W_i^1, W_i^2, b_i), candidate cell state (W_C^1, W_C^2, b_C), output gate (W_o^1, W_o^2, b_o) and output (W_y *and* b_y). The training is based on the structure of the formula for gates and states. An LSTM circuit is represented through the following equations

$$f_t = \sigma(W_f^1.h_{t-1} + W_f^2.x_t + b_f) \tag{3.3.1}$$
$$i_t = \sigma(W_i^1.h_{t-1} + W_i^2.x_t + b_i) \tag{3.3.2}$$
$$\tilde{C}_t = tanh(W_C^1.h_{t-1} + W_C^2.x_t + b_C) \tag{3.3.3}$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{3.3.4}$$
$$o_t = \sigma(W_o^1.h_{t-1} + W_o^2.x_t + b_o) \tag{3.3.5}$$
$$h_t = o_t * tanh(C_t) \tag{3.3.6}$$
$$\hat{y}_t = f_y(W_y h_t + b_y) \tag{3.3.7}$$

Let $L_t(y_t, \hat{y}_t)$ be the loss function at lag t. The total loss function is defined as

$$L = \sum_t L_t(y_t, \hat{y}_t).$$

where y_t is the real output and $\hat{y}_t$ is the predicted output of the LSTM network. To reach our goal which is to update the weight matrices and bias vectors using a gradient-based algorithm, we must determine the gradient of the total loss function with respect to those weight matrices and bias vectors. To accomplish this task, we have to determine the gradient of $L_t(y_t, \hat{y}_t)$ for all t. The rest of the work in this section is focused on the process used to train LSTM algorithms. Considering $L_t(y_t, \hat{y}_t) = L_t$, we determine firstly

3.3.1 Derivative of the individual loss function with respect to the state($\tilde{C}_t$, h_t and C_t) Using chain rule.

$$\frac{\partial L_t}{\partial h_t} = \frac{\partial L_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t} \tag{3.3.8}$$
$$\frac{\partial L_t}{\partial \tilde{C}_t} = \frac{\partial L_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\frac{\partial h_t}{\partial C_t}\frac{\partial C_t}{\partial \tilde{C}_t} \tag{3.3.9}$$
$$\frac{\partial L_t}{\partial C_t} = \frac{\partial L_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\frac{\partial h_t}{\partial C_t} \tag{3.3.10}$$

3.3.2 Derivative of the loss function with respect to the gates (f_t, i_t and o_t) Using chain rule.

$$\frac{\partial L_t}{\partial o_t} = \frac{\partial L_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\frac{\partial h_t}{\partial o_t} \tag{3.3.11}$$

$$\frac{\partial L_t}{\partial i_t} = \frac{\partial L_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\frac{\partial h_t}{\partial C_t}\frac{\partial C_t}{\partial i_t} = \frac{\partial L_t}{\partial C_t}\frac{\partial C_t}{\partial i_t} \tag{3.3.12}$$

$$\frac{\partial L_t}{\partial f_t} = \frac{\partial L_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\frac{\partial h_t}{\partial C_t}\frac{\partial C_t}{\partial f_t} = \frac{\partial L_t}{\partial C_t}\frac{\partial C_t}{\partial f_t} \tag{3.3.13}$$

3.3.3 Derivative of the loss function with respect to the weights (f_t, i_t and o_t) using the chain rule . Having the derivative of the individual loss function L_t with respect to the state and the gate, it is quite easy to determine the derivative of the L_t with respect to the weights and biases by using the Chain rules.

For instance, with the forget gate's weights and biases, we have

$$\frac{\partial L_t}{\partial W_f^1} = \frac{\partial L_t}{\partial f_t}\frac{\partial f_t}{\partial W_f^1} \tag{3.3.14}$$

$$\frac{\partial L_t}{\partial W_f^2} = \frac{\partial L_t}{\partial f_t}\frac{\partial f_t}{\partial W_f^2} \tag{3.3.15}$$

$$\frac{\partial L_t}{\partial b_f} = \frac{\partial L_t}{\partial f_t}\frac{\partial f_t}{\partial b_f} \tag{3.3.16}$$

Remember we have computed $\frac{\partial L_t}{\partial f_t}$ in eq.(3.3.13)

We process identically to determine the derivative of L_t with respect to the weights and biases into the rest of the gate and state.

3.3.4 Derivative of the total loss function with respect to the weights matrices and biases vector . The derivative of the total loss function with respect to the weight matrices and bias vector is determined using the fact that the derivative of a summation of continuous functions is the equation of the summation of the derivative of these functions. Therefore, we determine the derivative of L with respect to all weight matrices and biases using the following formula

$$\frac{\partial L}{\partial W} = \frac{\partial}{\partial W}\sum_t L_t = \sum_t \frac{\partial L_t}{\partial W}.$$

Afterward, all weight matrices and bias vectors are updated using optimization algorithms such as gradient descent algorithms.

3.4 Variant of LSTM

Several variants of LSTM exist in the literature but we will focus on only the Peephole LSTM and Convolutional LSTM and in the next section we will be talking about bidirectional-LSTM.

3.4.1 Peephole LSTM. Peephole LSTM has the same structure as normal LSTM, the difference between them is that in Peephole LSTM all the gate layers have access to the previous cell state or memory C_{t-1} (Yang et al., 2018). Peephole LSTM follows the same process as LSTM as it is described in the subsection Long Short-Term Memory (LSTM) Principle. The equation of the gates and states that govern peephole LSTM are the following

$$f_t = \sigma(W_f^1.x_t + W_f^2.h_{t-1} + W_f^3.C_{t-1} + b_f) \tag{3.4.1}$$
$$i_t = \sigma(W_i^1.x_t + W_i^2.h_{t-1} + W_i^3.C_{t-1} + b_i) \tag{3.4.2}$$
$$\tilde{C}_t = tanh(W_c^1.x_t + W_c^2.h_{t-1} \tag{3.4.3}$$
$$C_t = i_t.\tilde{C}_t + W_c^3.C_{t-1} + b_c) + C_{t-1}f_t \tag{3.4.4}$$
$$O_t = \sigma(W_o^1.x_t + W_o^2.h_{t-1} + W_o^3.C_{t-1} + b_o) \tag{3.4.5}$$
$$h_t = O_t * (C_t) \tag{3.4.6}$$
$$y_t = f_y(W_y h_t + b_y). \tag{3.4.7}$$

where (W_f^1, W_f^2,W_f^3 and b_f), (W_f^1, W_i^2, W_i^3 and b_i), (W_c^1, W_c^2 and W_c^3, b_c), (W_o^1 , W_o^2, W_o^3 and b_o) and (W_y and b_y) are respectively the weights matrices and bias vector of the forget gate, input gate, candidate cell state, output gate and output in the Peephole LSTM network.

3.4.2 Convolutional LSTM (ConvLSTM). The convolutional LSTM (ConvLSTM) is an LSTM version that is able to describe Spatio-temporal structures concurrently by directly incorporating spatial information into tensors. One of the characteristic that differ ConvLSTM from LSTM and classical LSTM is that all of the ConvLSTM's inputs $x_1, \ldots, x_t$, cell outputs $C_1, \ldots, C_t$, hidden states $h_1, \ldots, h_t$, and the gates i_t, f_t, O_t are 3D tensors reserving the last two dimensions as spatial dimensions (rows and columns), σ, σ_c and σ_g are activation functions. SHI et al. (2015). The dominating equation that governs ConvLSTM are

$$f_t = \sigma(W_f^1 * x_t + W_f^2 * h_{t-1} + W_f^3 \circ c_{t-1} + b_f) \tag{3.4.8}$$
$$i_t = \sigma(W_i^1 * x_t + W_i^2 * h_{t-1} + W_i^3 \circ c_{t-1} + b_i) \tag{3.4.9}$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c^1 * x_t + W_c^2 * h_{t-1} + b_c) \tag{3.4.10}$$
$$o_t = \sigma_g(W_o^1 * x_t + W_o^2 * h_{t-1} + W_o^3 \circ c_t + b_o) \tag{3.4.11}$$
$$h_t = o_t \circ \sigma_h(c_t) \tag{3.4.12}$$

Note that * represents the convolution product and $\circ$ represents the element-wise product. As a multivariate time series (time series having several variables)

3.5 Bidirectional LSTM (Bi-LSTM)

3.5.1 Shortcomings of LSTM and Introduction to Bi-LSTM . The idea behind LSTM is to harness memory from previous experiences to improve predictions. However, knowing both past and future values of a variable may be useful since understanding the tale of a variable's realization can help you remedy the many mistakes you may make and LSTM does not have this system on its own.

3.5.2 Bi-LSTM principle. As the name implies, Bi-LSTM considers both information transmission: utilizing the past values to enhance the upcoming values, as well as using the next values to rectify the previous values. Bi-directional LSTM considers two distinct hidden layers. One hidden is called the forward hidden layer (L^{for}) and another hidden is called backward hidden (L^{back}) and aggregates the result of both layers that constitute the output of the algorithm. Assume you have a time series data $X = \{x_1, \cdots, x_T\}$, where T represents the last time step

Forward hidden layer

With the forward hidden layer, Bidirectional LSTM works as a normal LSTM. By generating $\forall$ $t \in \{1, \cdots, T\}$, the hidden state (h_t)and the memory cell (C_t), using the process describe in subsection Long Short-Term Memory (LSTM) Principle.

backward hidden layer

Simultaneously, Bidirectional LSTM has a layer that processes the data from the data at the last time step x_T to the one at the first time step x_1. This process generates another hidden state (h'_t)and the memory cell (C'_t).

result aggregation

The final result of bidirectional LSTM is obtained by aggregating $\forall$ $t \in \{1, \cdots, T\}$, h_t and h'_t via the formula

$$\hat{y}_t = Wh_t + W'h'_t + B_y \tag{3.5.1}$$

where W, W', B_y are respectively weight matrix from the forward hidden layer to the output, weight matrix from the backward hidden layer to the output, and bias vector.

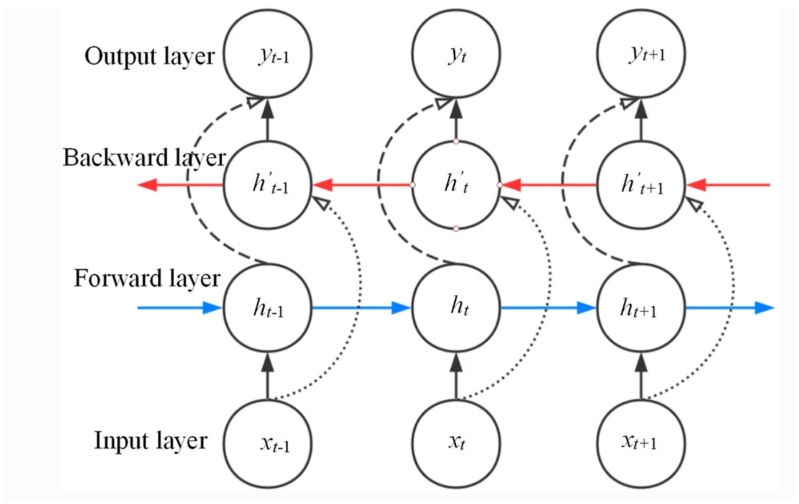

Figure 3.8: Bidirectional LSTM structure (Cheng et al., 2019)

3.5.3 Bi-LSTM training. Bi-LSTM uses gradient-based algorithms via backpropagation through time in order to update each weight matrices and bias vector of both the backward layer and forward layer.

3.6 Performance Metrics

Performance metrics are critical in machine learning since they determine whether or not a prediction is correct. Without performance measurements, one can not be certain of the correctness and efficiency of a model. It's critical to understand how your model interprets your data. There are other performance measures, but we will focus on four basic performance metrics and the information they give on model performance. In this section, we describe the performance metrics used to assess the prediction result of our models.

Assume $\hat{y}_t$ is the predicted value of the LSTM model and y_t is the real value of the variable we want to forecast at time step t.

3.6.1 Mean absolute error (MAE). The mean absolute error (MAE) is a measure of the difference in error between two observations reflecting the same phenomena. Comparisons of expected against observed, subsequent time versus starting time, one measuring technique versus another measurement technique, and so on. The MAE between $\hat{y}_t$ and y_t is computed as follows:

$$MAE(\hat{y}_t, y_t) = \frac{1}{T}\sum_{t=t}^{T}|y_t - \hat{y}_t| \tag{3.6.1}$$

3.6.2 Mean squared error (MSE). The mean squared error (MSE), also known as the mean squared deviation (MSD), determines the degree of inaccuracy in a statistical model (MSE) or in a prediction. in inference statistics, the MSE is a measure of similarity between a predicted value $\hat{y}_t$ and the real value y_t of a variable.

For a predictor $\hat{y}_t$, the MSE is defined as

$$MSE(\hat{y}_t, y_t) = \frac{1}{T}\sum_{t=t}^{T}(y_t - \hat{y}_t)^2 \tag{3.6.2}$$

3.6.3 Root mean squared error (RMSE). The root mean squared error (RMSE), is the square root of the (MSE).

For a predictor $\hat{y}_t$, the RMSE is defined as

$$RMSE(\hat{y}_t, y_t) = \sqrt{\frac{1}{T}\sum_{t=t}^{T}(y_t - \hat{y}_t)^2} \tag{3.6.3}$$

3.6.4 Mean absolute percentage error (MAPE). The mean absolute percentage error (MAPE) is a measure of a prediction system's accuracy. It reflects the average of each entry's absolute percentage error, indicating how accurate the forecasted values $\hat{y}_t$ were in relation to the actual values y_t. MAPE is sometimes useful for studying bigger collections of data, although it is impossible to calculate the MAPE of a variable having zero values. This is due to the computation requiring division by zero, which is not feasible.

$$MAPE(\hat{y}_t, y_t) = \frac{100\%}{T}\sum_{t=t}^{T}\left|\frac{y_t - \hat{y}_t}{y_t}\right| \tag{3.6.4}$$

4. Results and Discussion

In this part, we utilize LSTM and Bidirectional LSTM to forecast the variables temperature at 2 meters above sea level ($T2M$), wind speed at 2 meters ($WS2M$) and precipitation ($PREC$) in our data set. This chapter provides the data set description, the implementation procedure, the implementation outcomes, and a discussion of those results. The first and second sections explain the data set and the implementation procedure, the third displays the flow chart of the implementation, the fourth contains a fast learning curve analysis, the fifth section discusses about the prediction graphics obtained and finally the last section covers the performance metrics analysis of both the LSTM and Bi-LSTM methods.

4.1 Data Set

4.1.1 Data Set Information. The data set used in that study is a daily meteorological data set obtained from the following website: https://power.larc.nasa.gov/data-access-viewer/. The data set, containing 7 variables, represents the behavior over a period from **01/01/2015** to **12/31/2021** of 7 weather factors in Abidjan city (Ivory Coast) The size of this data is **(2557,7)** and its variables are

1. **T2M:** Temperature at 2 meters above the sea level ($°C$).
2. **WS2M:** Wind speed at 2 meters above the sea level (m/s).
3. **QV2M:** Specific Humidity at 2 meters above the sea level (g/kg)
4. **RH2M:** Relative Humidity at 2 meters above the sea level (%)
5. **PREC:** Precipitation (mm/day)
6. **SFRD:** Clear Sky Surface Shortwave Downward Irradiance ($kW-hr/m^2/day$)
7. **PS:** Surface Pressure (kPa)

4.1.2 Data Set Manipulation.

- Data transformation The first transformation applied to this data was to scale it using the command **MinMaxScaler()**, which normalizes each column of data based on its scale.

  Afterward, the data set was transformed into a supervised learning data set (data set having target variable and input variables). This conversion was completed using the sliding window method, by considering 14 days of observational data followed by a one-day prediction.

- Data splitting

After the data transformation, the data was divided into three parts: the train set, the validation set and the test set. This split was performed as follows: The first 1636 observations were assigned to training, the next 409 to validation, and the remaining 513, which correspond to the period "**2020-08-21 : 2021-12-31**", were reserved for testing the prediction skill of our models.

It should be noted that the train set is used to train the LSTM and Bi-LSTM models, the validation set is used to tweak their parameters and validate the model's performances and the test set is used to assess the accuracy of the prediction of LSTM and Bi-LSTM model.

4.1.3 Plot of the selected variables for forecasting. The selected variables for forecasting using LSTM and Bi-LSTM are $T2M$, $WS2M$ and $PREC$

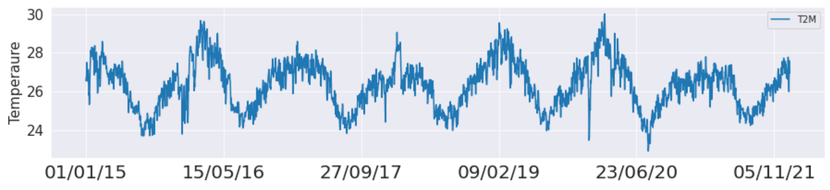

Figure 4.1: $T2M$ plot

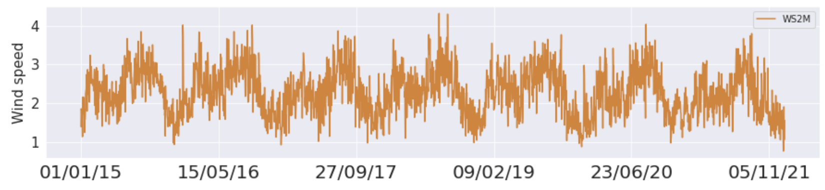

Figure 4.2: $WS2M$ plot

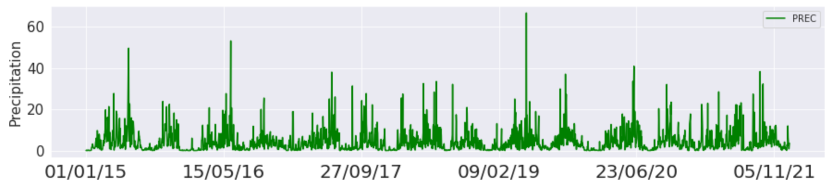

Figure 4.3: $PREC$ plot

Figures 4.1, 4.2 and 4.3 show respectively the plot of the variables $T2M$, $WS2M$ and $PREC$. We may see that the variables $T2M$ and $WS2M$ in figure 4.1 and 4.2 present a repeated pattern over time and have a low variability while the variable $PREC$ does not have a repeating pattern over time and presents a high variability.

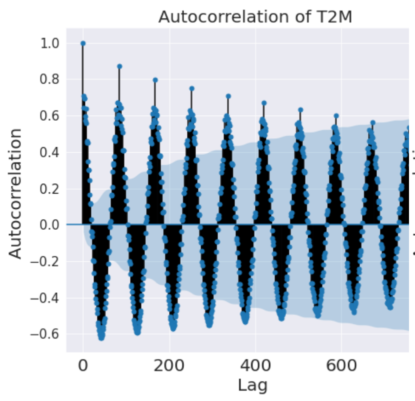

Figure 4.4: ACF plot of $T2M$

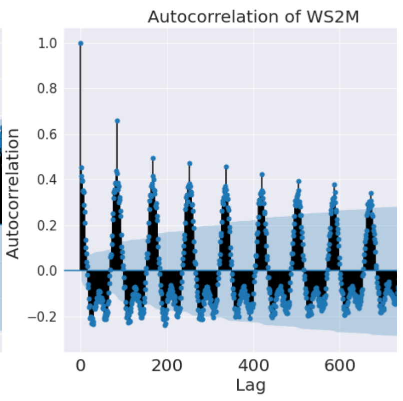

Figure 4.5: ACF plot of $WS2M$

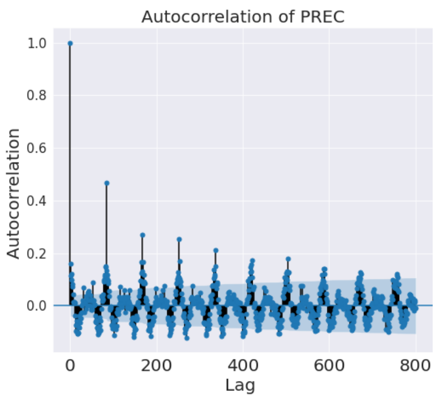

Figure 4.6: ACF plot of $PREC$

Figure 4.7: Auto-correlation plot of the variables $T2M$, $WS2M$ and $PREC$. The blue shaded zone in each plot represents the zone in which the correlation of the variable with its lag version is insignificant.

One can see through the figures 4.4, 4.5 and 4.6 that, the variables $T2M$, $WS2M$, and $PREC$ are at some extents, correlated to their lag, sometimes positively and sometimes negatively. But the auto-correlation of all variables is decreasing. This means that the nearest past observations of these variables have a bigger effect on them than the furthest observations. The auto-correlation plot of $T2M$ displays a sinusoidal variation, with a period of 40 lags, which means that when the variable $T2M$ increases the first day, 40 days later it decreases and vice versa. The auto-correlation of $WS2M$ shows almost a sinusoidal curve with a period of 20 lags but each negative variation, contains a sub-fluctuation having a period of 15 lags. This indicates that when $WS2M$ rises on the first day, 20 days later, and 15 days later, it will drop with the same amplitude as before, and vice versa. The variable $PREC$ presents a positive correlation for each period of 80 lags and a negative correlation for every 20 lags. This means that as $PREC$ increases the first day, 20 days later it will decrease while 100 days later it will increase again, and vice versa. However, from certain lag, the correlation between the variables and their lagged version starts to be nonsignificant. For $T2M$, the correlation is not significant from lag 490, for $WS2M$, the negative correlation is non-significant from lag 350 while the positive correlation stay significant but keep decreasing and the variable $PREC$ start to have a nonsignificant positive correlation from lag 780 and a non-significant negative correlation from lag 450.

4.2 LSTM and Bi-LSTM Implementation

This section describes the step by step implementation of our models to forecast the variables temperature at 2 meters from the sea level ($T2M$), precipitation ($PREC$), and wind speed at 2 meters ($WS2M$). The given code in the section Appendix is inspired by (srivatsan, 2020) and (LLC, 2022)

4.2.1 Environment. The implementation has been done on google colab. Note that google colab is an executable document that allows you to develop, run, and share code within Google Drive. Colab is in reality, a Jupyter notebook saved on Google Drive.

4.2.2 Some Libraries utilized.

1. **Pandas:** Pandas is an open-source Python library that offers powerful data manipulation techniques in Python. It is one of the most often used libraries in Python when working with data since its primary role is data administration and analysis. in python.

2. **Numpy:** NumPy, which stands for Numerical Python, is an open-source library developed in 2005 by Travis Oliphant. Numpy is responsible for all mathematical operator and logical operations in python and gives access to mathematical tools needed for array manipulation.

3. **Matplotlib:** Matplotlib is a Python on open-source data visualization and graphical plotting library that was created by John D. Hunter. It is used for producing stationary, dynamic, and interactive visualizations.

4. **Scikit-learn:** Scikit-learn is a machine learning library that is open source and handles both supervised and unsupervised learning. It also includes tools for model fitting, data preprocessing, model selection, model assessment, and a variety of additional functions.

5. **Keras:** Keras is a lightweight Python deep learning library that may be used with TensorFlow. It was created to make deep learning model implementation as quick and simple as feasible for research and development. Given the underlying frameworks, it can operate on Python 2.7 or 3.5 and easily execute on GPUs and CPUs. It has been developed by a Google team and especially by Francois Chollet. It's available under the permissive MIT license.

4.2.3 Step of the implementation.

1. Importation of the needed libraries

   We import the necessary libraries in order to implement the model.

2. Defining the LSTM and Bi-LSTM models in Keras

   The model is primarily constructed by describing the number of neurons in the hidden layer, the input size, whether we return the hidden state or not, an activation function, whether we allow certain neurons to be dropped out, and so on. It should be noted that the parameters fixed in the model were determined by selecting the values that produced the best results.

3. Compilation and training of the LSTM and Bi-LSTM model.

   The compilation phase refers to the initialization of the computing parameters of our models, where we fix the bases of the optimization methods used for training the models. Note that for both method (LSTM and Bi-LSTM) was trained using "**adam**" optimizer and "**relu**" activation function. This step may be seen as a pre-computing phase of the model.

After compiling the model, it is ready to be trained by modifying the different weights in the network using an optimization-based gradient method via backpropagation to suit the training data. The validation set is then used to validate the model outcome. The command **fit()** is used to train the model, with arguments including the train set, the epoch number on which we selected to train our model, the loss function used for optimization, and so on.

4. Model trained evaluation

   This step consists of evaluating the model via performance metrics using the validation set through the command **evaluate()**

5. Model Prediction

   The model predictions are made at this stage. Here the input test set scaled is introduced into the model, which gives the predicted values according to the input. This is done by using the command **predict()**. These predicted values were scaled so we used the function **inverse_transform()** to give its the real scale.

6. Prediction evaluation using performance metrics

   This is the last step of the implementation, where we use some performance metrics to assess the quality of the LSTM prediction.

4.3 Flow Chart of the Implementation

Figure 4.8 represents the flow chart of our implementation. It describes the step of our LSTM implementation.

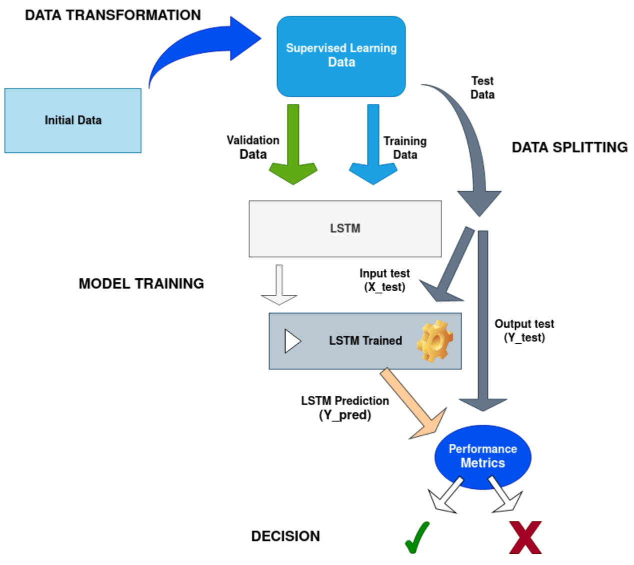

Figure 4.8: Flow chart of the implementation

4.4 Learning Curves

This section shows the learning curves of Bi-LSTM and LSTM for temperature, wind speed, and precipitation prediction. Note that, in all learning curve graphics, the evaluation metrics chosen were the mean absolute error. Keep in mind that (mean_absolute_error) represents the curve of the error during training and (val_mean_absolute_error) represents the curve of the error during validation.

4.4.1 LSTM and Bi-LSTM learning curves for temperature prediction. Figures 4.9 and 4.10 display the LSTM and Bi-LSTM learning curves for temperature prediction. The LSTM errors (training and validation) are decreasing quickly from the starting epoch to epoch 90 and decrease slightly after epoch 80 While Bi-LSTM errors (training and validation) decrease quickly from the beginning up to epoch 30 after they (errors) seem to be constant over epoch.

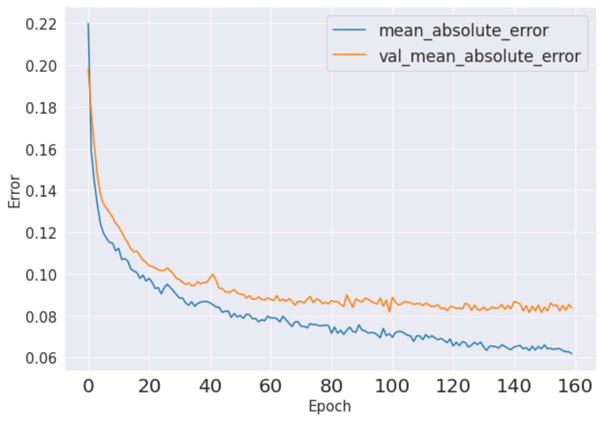

Figure 4.9: LSTM Learning curves for temperature prediction

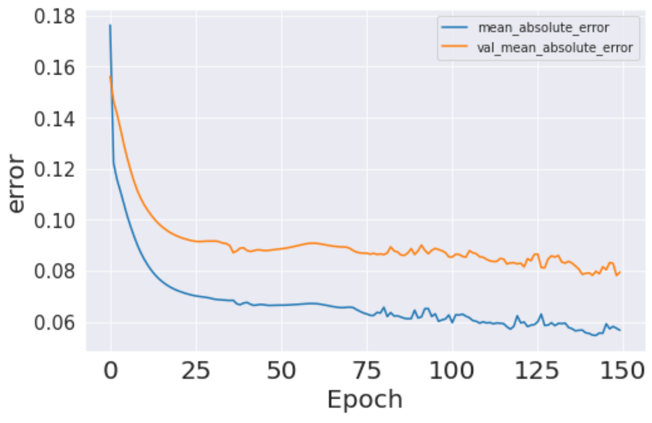

Figure 4.10: Bi-LSTM Learning curves for temperature prediction

4.4.2 LSTM and Bi-LSTM learning curves for wind speed prediction. Figures 4.11 and 4.12 shows respectively the learning curves of LSTM and Bi-LSTM for wind speed prediction. Both algorithms have been trained over 80 epochs and their errors (training and validation) are decreasing in both cases from epoch 40, LSTM validation error becomes alike constant while its training error decreases, and Bi-LSTM errors (training and validation) seem to be constant. Both algorithms seem to be a good fit for the data but Bi-LSTM shown to be the best fit.

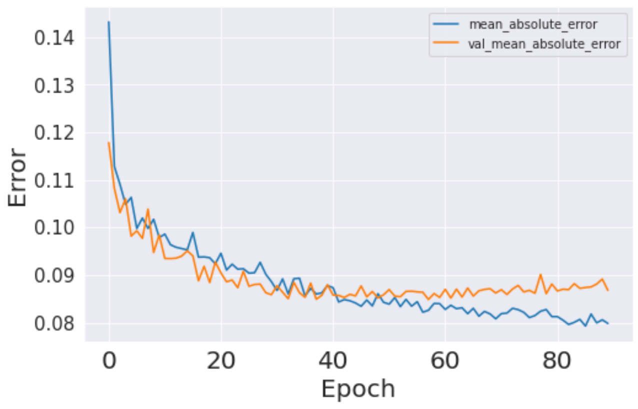

Figure 4.11: LSTM learning curves for wind speed prediction

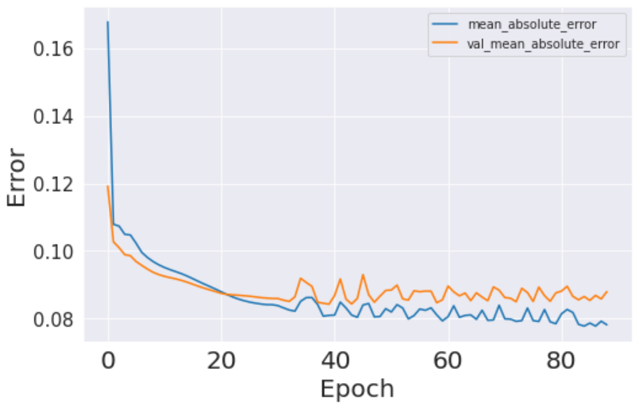

Figure 4.12: Bi-LSTM learning curves for wind speed prediction

4.4.3 LSTM and Bi-LSTM learning curves for precipitation prediction. Figures 4.13 and 4.14 present the respective learning curves of LSTM and Bi-LSTM. LSTM error curves have several little fluctuations and the training error decreases along with the epoch while the validation error decrease by epoch 50 and after seems to increase until the end. Bi-LSTM errors are a bit smooth, they decrease after getting a constant value over epochs but from epoch 80 the curve passes from smooth to rough. For both Bi-LSTM and LSTM algorithms, the validation error stay generally below the training error. This could be a sign that the validation test is easier to be learned than the train set.

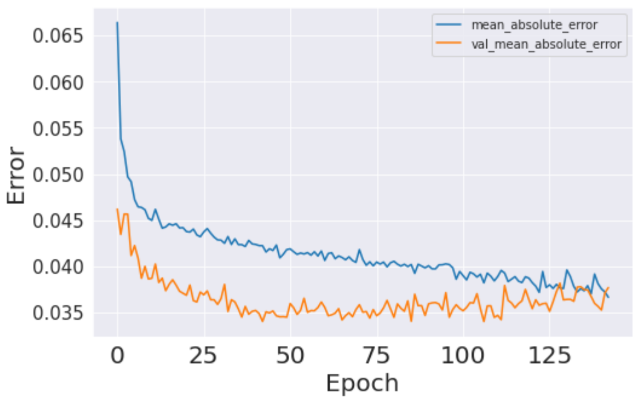

Figure 4.13: LSTM learning curve for precipitation prediction

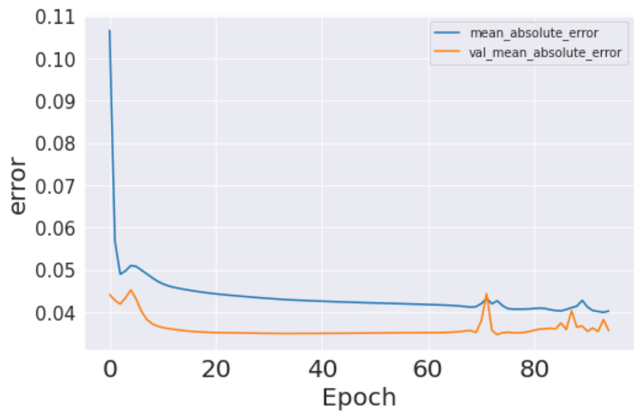

Figure 4.14: Bi-LSTM learning curve for precipitation prediction

4.5 Forecasting Visualization

This part is focused on the visualization of predicting results. Note that, the LSTM and Bi-LSTM predictions have been done over the variables $T2M$, $WS2M$, and $PREC$. It should be noted that the orange lines in each figure represent the actual values of the variable under examination, whilst the green lines represent the predicted values of the model under study.

4.5.1 LSTM for the temperature at 2 meters above the sea level prediction (T2M). Figure 4.15 depicts a plot of the LSTM prediction of the variable $T2M$ and its true values. In this figure, we can see that the forecast and the actual values of the variable $T2M$ are nearly identical, demonstrating how accurate the LSTM prediction is. However, some variations of $T2M$, above all the extreme variation, were difficult to be captured by LSTM. This fact means that the seasonality of the variable $T2M$ was mostly learned using LSTM but when the temperature at 2 meters above the sea level hits some unexpected values, LSTM fails to predict correctly.

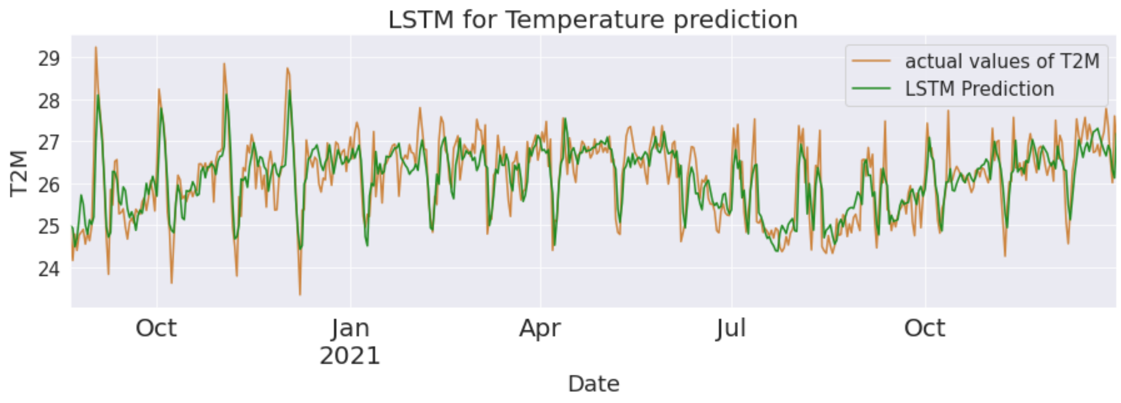

Figure 4.15: LSTM prediction of T2M vs Actual values

4.5.2 LSTM for wind speed at 2 meters above the sea level prediction (WS2M). Figure 4.16 shows the comparison between actual values and LSTM prediction of the variable $WS2M$. In that figure, the LSTM forecast seems to have the same variation as the actual observed values of $WS2M$ over time; but several of its fluctuations were not well captured by LSTM. Therefore, one can affirm that when LSTM was predicting $WS2M$, it learned correctly the seasonality and the general pattern of the variable $WS2M$ but not exactly its fluctuation amplitudes.

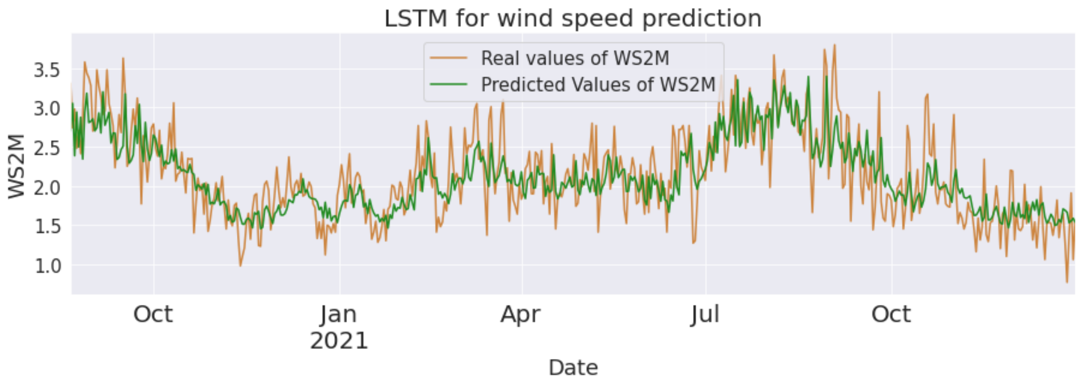

Figure 4.16: LSTM prediction of WS2M vs Actual values

4.5.3 LSTM for precipitation (PREC) prediction . In figure 4.17, we might see both the LSTM prediction value and the real value of the variable precipitation in the same graph. This graph shows a significant difference between prediction and real observation values of $PREC$, showing how inaccurate was the LSTM prediction. The main difference between LSTM prediction and the real observation values of $PREC$ is in their respective peak values. While on average the peak of the actual precipitation observation is approximately 30 mm/day, the anticipated value of the LSTM is around 10 mm/day. This inability of LSTM to predict the variable $PREC$ is most likely due to its high variability and lack of a recurrent pattern in its variation.

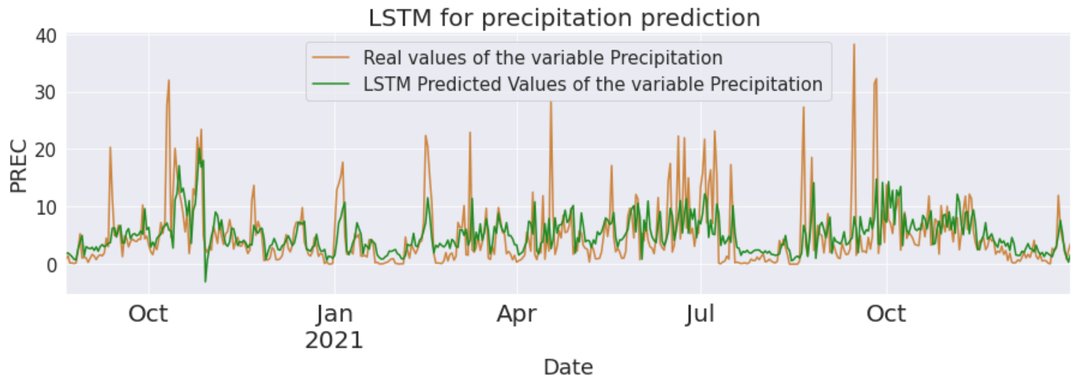

Figure 4.17: LSTM prediction of PREC vs Actual values

4.5.4 Bi-LSTM for the temperature at 2 meters above the sea level prediction (T2M). In figure 4.18, is shown a graph containing both plots of the Bi-LSTM prediction of $T2M$ and its real values. In that figure, we may notice first of all that Bi-LSTM shows a little improvement compared to LSTM. Secondly, it is worth emphasizing that the Bi-LSTM prediction result of the variable $T2M$ is relatively good however it has a difficulty in forecasting the severe temperatures and present sometimes a delay in $T2M$ prediction. The difference between actual and Bi-LSTM predicted values is mainly seen for the period "Aug 2021 - Jan 2021" where the real observations of temperature reach its biggest and lowest values. Like LSTM prediction, Bi-LSTM primarily learned the repeated pattern of the variable over time but not exactly the individual fluctuation.

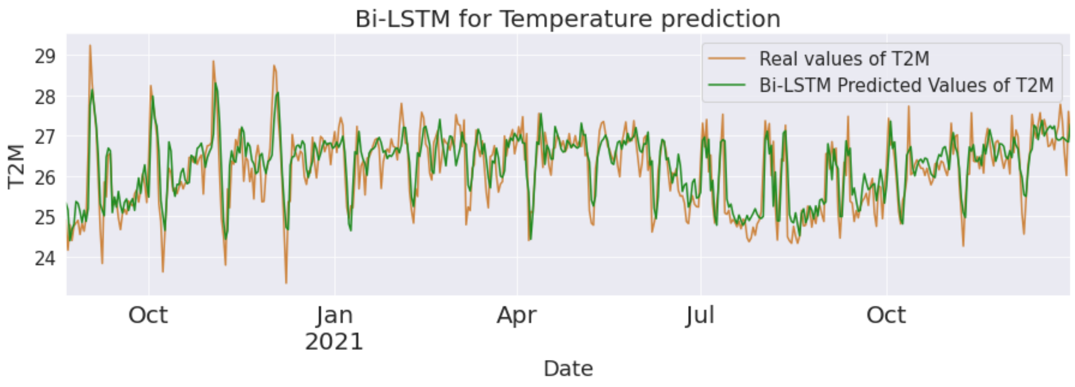

Figure 4.18: Bi-LSTM prediction of T2M vs Actual values

4.5.5 Bi-LSTM for wind speed at 2 meters above the sea level prediction (WS2M). Figure 4.18 shows a comparison plot of Bi-LSTM prediction and actual values of the variable $WS2M$. In comparison to LSTM, we see an improvement in Bi-LSTM forecasting for the variable $WS2M$. The Bi-LSTM predicted values and real $WS2M$ observations are almost similar. However, the difference between them is most obvious in the Bi-LSTM prediction of the extreme observation values of $WS2M$, where Bi-LSTM struggles to accurately track its variation.

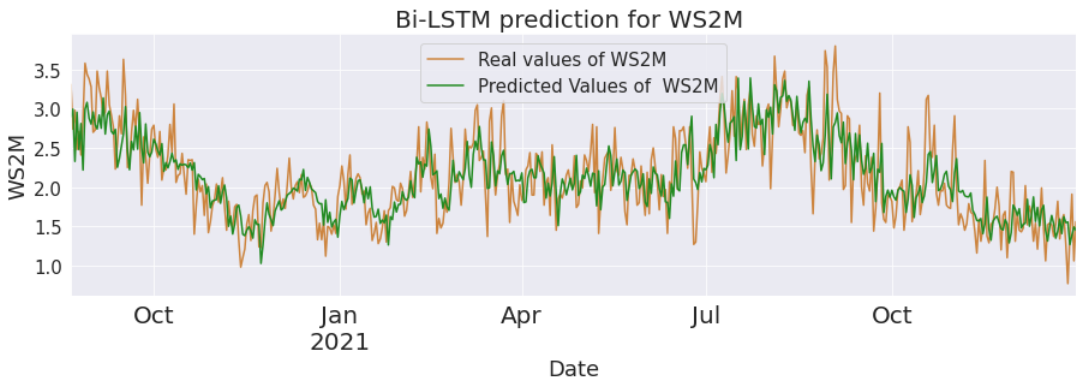

Figure 4.19: Bi-LSTM prediction of WS2M vs Actual values

4.5.6 Bi-LSTM for precipitation (PREC) prediction . The forecast is not especially accurate, as shown in Figure 4.20, since the Bi-LSTM prediction curve does not reach in general, the peak values of the real observation of $PREC$. Yet, at low amplitude levels, Bi-LSTM prediction was somewhat similar to the true value, which makes Bi-LSTM prediction better than that of LSTM. This fact indicates that Bi-LSTM is unable of dealing with the high levels of fluctuation of variable $PREC$.

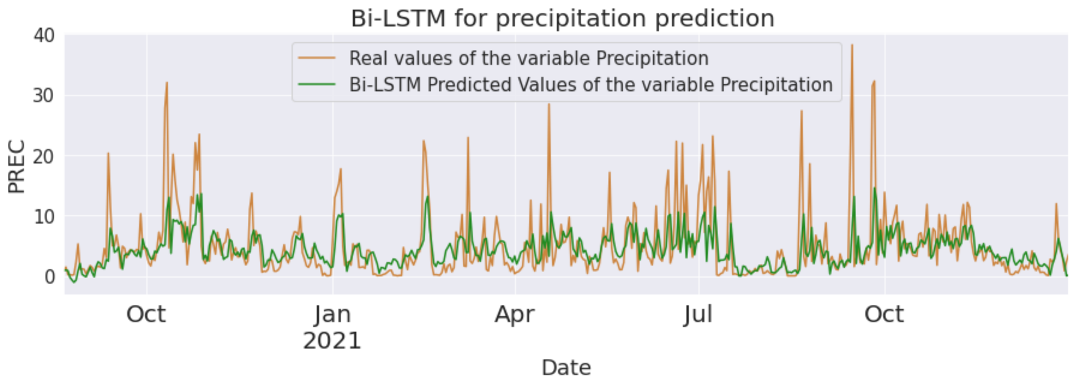

Figure 4.20: Bi-LSTM prediction of PREC vs actual values

4.6 Performance Metrics Analysis

This part is reserved for the performance metrics results for both LSTM and BI-LSTM for the variables $T2M$, $WS2M$, and $PREC$.

Model	Weather Component	MSE	MAE	MAPE
LSTM	Temperature (T2M)	0.319	0.431	0.016
	Wind speed (WS2M)	0.134	0.285	0.141
	Precipitation (PREC)	23.463	3.066	%
Bi-LSTM	Temperature (T2M)	0.313	0.43	0.017
	Wind speed (WS2M)	0.128	0.279	0.139
	Precipitation (PREC)	23.769	2.902	%

Table 4.1: Performance Metrics of LSTM and Bi-LSTM

Note that we will not compute the MAPE for the variable *PREC* since it takes sometimes the value 00 and the MAPE does not handle this kind of variable, see equation (3.6.4).

MSE and MAE

As the table 4.1 indicates, the best MSE and MAE are seen in temperature $T2M$ and wind speed $WS2M$ prediction where Bi-LSTM outperformed LSTM in general. The $WS2M$ prediction gives the best MSE and MAE results of the table 4.1 in both LSTM and Bi-LSTM prediction and the poorest MSE and MAE are seen in $PREC$ prediction.

MAPE

MAPE determine in average, the percentage of mean absolute error done, for each values predicted by our algorithms. Table 4.1 indicates that both LSTM and Bi-LSTM show that all MAPE results are below 18% which is a good performance. LSTM indicates 1.6% percent of MAPE in temperature, while it displays 14% for wind speed prediction. Bi-LSTM has a MAPE of 1.7% in temperature prediction while it has 13.9% in wind speed prediction. We might see that Bi-LSTM outperformed LSTM in MAPE while predicting the variables $T2M$ however, LSTM's MAPE was slightly better than that of Bi-LSTM for predicting $WS2M$. The best MAPE score of both algorithms is recorded in $T2M$ prediction. In other word, Bi-LSTM and LSTM made less errors when they have predicted $T2M$ than in $WS2M$ prediction.

5. Conclusion and Future Work

This paper proposes the use of LSTM and Bi-LSTM techniques to model and predicts weather components such as temperature at two meters ($T2M$), wind speed at two meters ($WS2M$), and precipitation ($PREC$) in Abidjan. It should be noted that the variables $T2M$ and $WS2M$ are seasonal and present a repeating pattern over time, while $PREC$ has a great degree of variability and does not have a recurring pattern in its variation. According to the forecasting result and performance metrics analysis, the use of LSTM and Bi-LSTM for one-day forecasting yielded rather accurate predictions of temperature and wind speed; however, both methods fail to forecast the extreme values of these variables and perform poorly in precipitation prediction. It is correct to claim that LSTM and Bi-LSTM effectively predicted the variables $T2M$ and $WS2M$ due to their seasonal aspect and regular variation over time, but also failed to predict the variable $PREC$ due to its high variability and lack of recurring pattern over time. As a consequence, one can say that the prediction results of both techniques are more reliable when the variables they forecast are seasonal and do not take extreme values; nevertheless, their prediction is less reliable when the variables have high variability and do not indicate seasonality in their variation. It is worth emphasizing that these two methods, with minor improvements on their own, would be quite effective and deserve full use in weather forecasting. Finally, it is critical to recognize that deep learning is a major and valuable tool in weather prediction. One of the first good characteristics of deep learning tools is the ease of use, and as we showed in that essay, deep learning algorithms may effectively predict variables with few outliers. Therefore, a potential study field may be the use of deep learning techniques, namely LSTM approaches, in prediction employing an outlier management system.

References

The nobel prize in physics 2021. URL https://www.nobelprize.org/prizes/physics/2021/popular-information/.

Alexander Amini. Introduction to deep learning. *MIT*, 6:S191, 2019.

Richard L Bankert. Cloud classification of avhrr imagery in maritime regions using a probabilistic neural network. *Journal of Applied Meteorology and climatology*, 33(8):909–918, 1994.

Peter Bauer, Alan Thorpe, and Gilbert Brunet. The quiet revolution of numerical weather prediction. *Nature*, 525(7567):47–55, 2015.

John Billet, Mark DeLisi, Brian G Smith, and Cory Gates. Use of regression techniques to predict hail size and the probability of large hail. *Weather and Forecasting*, 12(1):154–164, 1997.

Vilhelm Bjerknes, Esther Volken, and S Bronnimann. The problem of weather prediction, considered from the viewpoints of mechanics and physics. *Meteorologische Zeitschrift*, 18(6):663, 2009.

Bogdan Bochenek and Zbigniew Ustrnul. Machine learning in weather prediction and climate analyses—applications and perspectives. *Atmosphere*, 13(2):180, 2022.

John Bjørnar Bremnes. Probabilistic forecasts of precipitation in terms of quantiles using nwp model output. *Monthly Weather Review*, 132(1):338 – 347, 2004. doi: 10.1175/1520-0493(2004)132⟨0338:PFOPIT⟩2.0.CO;2. URL https://journals.ametsoc.org/view/journals/mwre/132/1/1520-0493_2004_132_0338_pfopit_2.0.co_2.xml.

Sean D Campbell and Francis X Diebold. Weather forecasting for weather derivatives. *Journal of the American Statistical Association*, 100(469):6–16, 2005. doi: 10.1198/016214504000001051. URL https://doi.org/10.1198/016214504000001051.

J. G. Charney, R. FjÖrtoft, and J. Von Neumann. Numerical integration of the barotropic vorticity equation. *Tellus*, 2(4):237–254, 1950. doi: 10.3402/tellusa.v2i4.8607. URL https://doi.org/10.3402/tellusa.v2i4.8607.

Rui-Bin Chen, Xiang Wang, Weiming Zhang, Xiaoyu Zhu, Aiping Li, and Chao Yang. A hybrid cnn-lstm model for typhoon formation forecasting. *GeoInformatica*, pages 1–22, 2019.

Hongju Cheng, Zhe Xie, Leihuo Wu, Zhiyong Yu, and Ruixing Li. Data prediction model in wireless sensor networks based on bidirectional lstm. *EURASIP Journal on Wireless Communications and Networking*, 2019(1):1–12, 2019.

Vincent Davis-Reddy and Mambo. Socio-economic impacts of extreme weather events in southern africa. *Climate Risk and Vulnerability: A Handbook for Southern Africa, pp. 30-47*, 2017.

Saul Dobilas. Lstm recurrent neural networks-how to teach a network to remember the past, Mar 2022. URL https://towardsdatascience.com/lstm-recurrent-neural-networks-how-to-teach-a-network-to-remember-the-past-55e54c2ff22e.

Jaroslav Frnda, Marek Durica, Jan Nedoma, Stanislav Zabka, Radek Martinek, and Michal Kostelansky. A weather forecast model accuracy analysis and ecmwf enhancement proposal by neural network. *Sensors*, 19(23), 2019. ISSN 1424-8220. doi: 10.3390/s19235144. URL https://www.mdpi.com/1424-8220/19/23/5144.

Tilmann Gneiting, Adrian E. Raftery, Anton H. Westveld, and Tom Goldman. Calibrated probabilistic forecasting using ensemble model output statistics and minimum crps estimation. *Monthly Weather Review*, 133(5):1098 – 1118, 2005. doi: 10.1175/MWR2904.1. URL https://journals.ametsoc.org/view/journals/mwre/133/5/mwr2904.1.xml.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

Sue Ellen Haupt, Antonello Pasini, and Caren Marzban. *Artificial intelligence methods in the environmental sciences*. Springer Science & Business Media, 2008.

Pradeep Hewage, Ardhendu Behera, Marcello Trovati, and Ella Pereira. Long-Short Term Memory for an Effective Short-Term Weather Forecasting Model Using Surface Weather Data. In John MacIntyre, Ilias Maglogiannis, Lazaros Iliadis, and Elias Pimenidis, editors, *15th IFIP International Conference on Artificial Intelligence Applications and Innovations (AIAI)*, volume AICT-559 of *Artificial Intelligence Applications and Innovations*, pages 382–390, Hersonissos, Greece, May 2019. Springer International Publishing. doi: 10.1007/978-3-030-19823-7_32. URL https://hal.inria.fr/hal-02331313. Part 7: Deep Learning - Convolutional ANN.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9: 1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

Lars Kamer. Africa: Types of reported natural disasters, Feb 2022. URL https://www.statista.com/statistics/1270517/distribution-of-reported-natural-disasters-in-africa-by-type/.

AKIRA KASAHARA and WARREN M. WASHINGTON. Ncar global general circulation model of the atmosphere. *Monthly Weather Review*, 95(7):389 – 402, 1967. doi: 10.1175/1520-0493(1967)095⟨0389:NGGCMO⟩2.3.CO;2. URL https://journals.ametsoc.org/view/journals/mwre/95/7/1520-0493_1967_095_0389_nggcmo_2_3_co_2.xml.

Hideji Kida, Takashi Koide, Hidetaka Sasaki, and Masaru Chiba. A new approach for coupling a limited area model to a gcm for regional climate simulations. *Journal of the Meteorological Society of Japan. Ser. II*, 69(6):723–728, 1991.

William H Klein, Billy M Lewis, and Isadore Enger. Objective prediction of five-day mean temperatures during winter. *Journal of Atmospheric Sciences*, 16(6):672–682, 1959.

Shuo Liang, Dingcheng Wang, Jingrong Wu, Rui Wang, and Ruiqi Wang. Method of bidirectional lstm modelling for the atmospheric temperature. *Intelligent Automation Soft Computing*, 29: 701–714, 01 2021. doi: 10.32604/iasc.2021.020010.

MultiMedia LLC. Time series prediction with lstm recurrent neural networks in python with keras. machine learning mastery., 2022. URL https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/.

Alicia Lozano-Diez, Ruben Zazo, Doroteo Toledano, and Joaquin Gonzalez-Rodriguez. An analysis of the influence of deep neural network (dnn) topology in bottleneck feature based language recognition. *PLOS ONE*, 12:e0182580, 08 2017. doi: 10.1371/journal.pone.0182580.

Thomas F Malone. Application of statistical methods in weather prediction. *Proceedings of the National Academy of Sciences of the United States of America*, 41(11):806, 1955.

Christopher Olah. Understanding lstm networks, 2015. URL https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Yashon O Ouma, Rodrick Cheruyot, and Alice N Wachera. Rainfall and runoff time-series trend analysis using lstm recurrent neural network and wavelet neural network with satellite-based meteorological data: case study of nzoia hydrologic basin. *Complex & Intelligent Systems*, 8 (1):213–236, 2022.

Norman A. Phillips. The general circulation of the atmosphere: A numerical experiment. *Quarterly Journal of the Royal Meteorological Society*, 82(352):123–164, 1956. doi: 10.1002/qj.49708235202.

Roger A Pielke Sr, Toshihisa Matsui, Giovanni Leoncini, Timothy Nobis, Udaysankar S Nair, Er Lu, Joseph Eastman, Sujay Kumar, Christa D Peters-Lidard, Yudong Tian, et al. A new paradigm for parameterizations in numerical weather prediction and other atmospheric models. *National Weather Digest*, 30:93–99, 2006.

Donlapark Ponnoprat. Short-term daily precipitation forecasting with seasonally-integrated autoencoder, 01 2021.

Adrian E. Raftery, Tilmann Gneiting, Fadoua Balabdaoui, and Michael Polakowski. Using bayesian model averaging to calibrate forecast ensembles. *Monthly Weather Review*, 133(5):1155 – 1174, 2005. doi: 10.1175/MWR2906.1. URL https://journals.ametsoc.org/view/journals/mwre/133/5/mwr2906.1.xml.

Xiaoli Ren, Xiaoyong Li, Kaijun Ren, Junqiang Song, Zichen Xu, Kefeng Deng, and Xiang Wang. Deep learning-based weather prediction: a survey. *Big Data Research*, 23:100178, 2021.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Jakob Runge, V. Petoukhov, Jonathan Donges, Jaroslav Hlinka, Nikola Jajcay, Martin Vejmelka, David Hartman, Norbet Marwan, Milan Palus, and Juergen Kurths. Identifying causal gateways and mediators in complex spatio-temporal systems. *Nature Communications*, 6:8502, 10 2015. doi: 10.1038/ncomms9502.

Sebastian Scher. Toward data-driven weather and climate forecasting: Approximating a simple general circulation model with deep learning. *Geophysical Research Letters*, 45, 11 2018. doi: 10.1029/2018GL080704.

MG Schultz, Clara Betancourt, Bing Gong, Felix Kleinert, Michael Langguth, LH Leufen, Amirpasha Mozaffari, and Scarlet Stadtler. Can deep learning beat numerical weather prediction? *Philosophical Transactions of the Royal Society A*, 379(2194):20200097, 2021.

Alex Sherstinsky. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *CoRR*, abs/1808.03314, 2018. URL http://arxiv.org/abs/1808.03314.

Xingjian SHI, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun WOO. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/file/07563a3fe3bbe7e3ba84431ad9d055af-Paper.pdf.

Xingjian Shi, Zhihan Gao, Leonard Lausen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun Woo. Deep learning for precipitation nowcasting: A benchmark and a new model, 2017. URL https://arxiv.org/abs/1706.03458.

GJ Shutts and TN Palmer. Convective forcing fluctuations in a cloud-resolving model: Relevance to the stochastic parameterization problem. *Journal of climate*, 20(2):187–202, 2007.

S. srivatsan. End-to-end-time-series multivariate$_time_series_modeling_using_lstm.ipynbatmaster,$ 2020

Bin Wang, Jie Lu, Zheng Yan, Huaishao Luo, Tianrui Li, Yu Zheng, and Guangquan Zhang. Deep uncertainty quantification: A machine learning approach for weather forecasting, 2018. URL https://arxiv.org/abs/1812.09467.

Yunbo Wang, Mingsheng Long, Jianmin Wang, Zhifeng Gao, and Philip S Yu. Predrnn: Recurrent neural networks for predictive learning using spatiotemporal lstms. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/e5f6ad6ce374177eef023bf5d0c018b6-Paper.pdf.

Jonathan A. Weyn, Dale R. Durran, and Rich Caruana. Improving data-driven global weather prediction using deep convolutional neural networks on a cubed sphere. *Journal of Advances in Modeling Earth Systems*, 12(9):e2020MS002109, 2020. https://doi.org/10.1029/2020MS002109. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2020MS002109. e2020MS002109 10.1029/2020MS002109.

Edmund P Willis and William H Hooke. Cleveland abbe and american meteorology, 1871–1901. *Bulletin of the American Meteorological Society*, 87(3):315–326, 2006.

Ting Yang, Huaizhi Wang, Saddam Aziz, Hui Jiang, and Jianchun Peng. A novel method of wind speed prediction by peephole lstm. pages 364–369, 11 2018. 10.1109/POWERCON.2018.8601550.

Chuanlong Yin, Yuefei Zhu, Jinlong Fei, and Xinzheng He. A deep learning approach for intrusion detection using recurrent neural networks. *IEEE Access*, 5:21954–21961, 2017. 10.1109/ACCESS.2017.2762418.

6. Appendix

```
###########Importation of needed libraries and package###########

import tensorflow as tf
import os
import pandas as pd
import numpy as np
import matplotlib as mpl
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.metrics import mean_squared_error
from keras.preprocessing.sequence import TimeseriesGenerator
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Bidirectional
from keras.layers import Dense, Input, LSTM, Embedding, Dropout, Activation
,TimeDistributed
import seaborn as sns
from keras.layers import Dense
from keras.layers import LSTM, GRU
import tensorflow as tf
from tensorflow import keras
from datetime import datetime
from keras import layers
from sklearn.metrics import mean_squared_error,mean_absolute_error
,r2_score,mean_absolute_percentage_error
from google.colab import files
mpl.rcParams['axes.grid'] = True
mpl.rcParams['xtick.labelsize'] = 20
mpl.rcParams['ytick.labelsize'] = 15
plt.rcParams['axes.labelsize'] = 20
plt.rcParams['axes.titlesize'] = 20
sns.set_style("darkgrid")
####Importation of Data##

df = pd.read_csv(r'/content/Daily Abidjan_data.csv')
```

```
                     ##############################
                     ####                      ####
                     #### Temperature (T2M)    ####
                     ####  prediction          ####
                     ##############################

#####################################################
###### LSTM for temperature (T2M)   prediction ######

#####################################################

########## Datapreprocessing For Temperature   #########
##### Sorting the index data #####
df_input= df[['T2M', 'WS2M', 'QV2M', 'RH2M', 'PREC', 'PS','SWRD']].sort_index()
#####  Scale the data sorted #####
scaler=MinMaxScaler()
data_scaled=scaler.fit_transform(df_input)
data=df_input.values


######define input data  and output data #########
features=data_scaled
target=data_scaled[:,0]
# features=data
# target=data[:,0]
##### Time series data transformation, construction of supervised learning data ####
TimeseriesGenerator(features, target,length=14,batch_size=1)[0]

 ###### Splitting the dataset into training set and test set #####

X_train,  y_train = features[:1636,:], target[:1636]
X_val,y_val= features[1636:2045,:], target[1636:2045]
X_test,y_test= features[2045:,:], target[2045:]

##### Train and test generator, definition
of the number of past
observations (window length) and the batch size. #####


window_length= 14
b_size= 124
features_num=7
train_generator= TimeseriesGenerator(X_train,y_train,length=window_length
,sampling_rate=1,batch_size=b_size)
```

```
val_generator= TimeseriesGenerator(X_val,y_val
,length=window_length,sampling_rate=1,batch_size=b_size)
test_generator= TimeseriesGenerator(X_test,y_test
,length=window_length,sampling_rate=1,batch_size=b_size)



##### Defining the LSTM model #####
model=Sequential()
model.add(LSTM(94, activation='relu',
input_shape=(window_length, features_num), return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(1))

model.summary()

###### Early stopping: Model stops if the
validation loss does not improve after
50 more iterations the algorithm stop#####

early_stopping= tf.keras.callbacks.EarlyStopping(monitor='val_loss'
,patience=50,mode='min')
model.compile(loss=tf.losses.MeanSquaredError()
,optimizer='adam'
,metrics=[tf.metrics.MeanAbsoluteError()
,tf.metrics.RootMeanSquaredError()])
history=model.fit(train_generator,epochs=160
,shuffle=False, validation_data=val_generator
,callbacks=[early_stopping])

##### Learning Curve plot ####
pd.DataFrame(history.history)[['loss','val_loss']].plot(figsize=(10,7))
plt.ylabel("Cost function",fontsize=15)
plt.xlabel("Epoch",fontsize=15)
plt.show()
pd.DataFrame(history.history)[['mean_absolute_error'
,'val_mean_absolute_error']].plot(figsize=(10,7))
plt.ylabel("Error",fontsize=15)
plt.xlabel("Epoch",fontsize=15)
plt.legend(fontsize=17)
plt.show()
```

```
###### Model Evaluation  #####
model.evaluate(test_generator,verbose=0)

###### Predictions #####

predictions=model.predict(test_generator)

##### Consider the values of X_test by ignoring the first 14th #####

New_X_test=X_test[:,1:][window_length:]

##### construction of a data set containing input test and LSTM prediction values
in order to apply inverse transform to give the real values
of the data after scaling #####

df_pred=pd.concat([pd.DataFrame(predictions)
,pd.DataFrame(New_X_test)],axis=1)

##### return the inverse transform
of the MinMaxscaler created above
to give the real value that we had initially####

rev_trans=scaler.inverse_transform(df_pred.values)

##### Extraction of real values vector and  prediction vector #####

df_final=df_input[-predictions.shape[0]:]
df1=df_final['T2M']
rev2=rev_trans[:,0]

##### Performance metrics ####

mse = mean_squared_error(df1, rev2)
r2 = r2_score(df1, rev2)
mae = mean_absolute_error(df1, rev2)
# mae_scaled = mean_absolute_error(X_test[-rev2.shape[0]:,0], predictions[:,0])
mape = mean_absolute_percentage_error(df1, rev2)
print("mse=",mse,3)
print("r2=",r2,3)
print("mae=",mae)
# print("mae_scaled=" + str(round(mae_scaled,2)))
print("mape=",mape)


##### Plot of the LSTM predicted values of T2M and actual values #####
```

```
dates = pd.date_range('08/21/2020', periods=498)
Forecast=pd.DataFrame({'actual values of T2M':df1,'LSTM Prediction ':rev2})
Forecast['date']=dates
For=Forecast.set_index('date')
# train_forecast[2000:2600].plot.line(figsize=(15, 7))
sns.set_style("darkgrid")
For.plot.line(figsize=(15, 4),color=['peru','forestgreen'])
plt.ylabel("T2M",fontsize=18)
plt.xlabel("Date",fontsize=18)
plt.title("LSTM for Temperature prediction")
plt.legend(fontsize=15)
plt.show()


##################################################
###### Bi-LSTM for temperature (T2M) prediction ####
#####################################################


######  Data Transformation ######

window_length= 14
b_size= 64
features_num=7
train_generatorp= TimeseriesGenerator(X_train
,y_train,length=window_length
,sampling_rate=1,batch_size=b_size)
val_generatorp= TimeseriesGenerator(X_val,y_val
,length=window_length
,sampling_rate=1,batch_size=b_size)
test_generatorp= TimeseriesGenerator(X_test,y_test
,length=window_length
,sampling_rate=1,batch_size=b_size)

model2= keras.Sequential()

model2.add(
    layers.Bidirectional
    (layers.LSTM(128
    , return_sequences=False), input_shape=(window_length
    , features_num))
)
model2.add(layers.Dense(1))
model2.summary()
#### Early stopping: Model stops
```

```
if the validation loss does
not improve after 50 more
iteration the algorithm stop #####

early_stopping1= tf.keras.callbacks.EarlyStopping(
monitor='val_loss',patience=50,mode='min')
model2.compile(loss=tf.losses.MeanSquaredError()
,optimizer='adam',metrics=[tf.metrics.MeanAbsoluteError()
,tf.metrics.RootMeanSquaredError()])
history2=model2.fit(train_generatorp
,epochs=150,shuffle=False, validation_data=val_generator
,callbacks=[early_stopping1])

##### Learning curve #####

pd.DataFrame(history2.history)[['loss','val_loss']]
.plot(figsize=(8,5))
plt.ylabel("Cost function")
plt.xlabel("Epoch")
plt.savefig('Bi-LSTM_TMP_LOSS.png')
files.download("Bi-LSTM_TMP_LOSS.png")
plt.show()
pd.DataFrame(history2.history)[['mean_absolute_error','val_mean_absolute_error']].plot(
plt.ylabel("error")
plt.xlabel("Epoch")
plt.savefig('Bi-LSTM_TMP_ERROR.png')
files.download("Bi-LSTM_TMP_ERROR.png")
plt.show()

###### Predictions #####

prediction21=model2.predict_generator(test_generator)
###### Consider the values of X_test by ignoring the first 14th#####

New_X_test2=X_test[:,1:][window_length:]

###### Concatenate Xtest and Y predicted for #####

df_pred12=pd.concat([pd.DataFrame(prediction21),
pd.DataFrame(New_X_test2)],axis=1)
###### return the inverse transform
of the MinMaxscaler created above to
give the real value that we had initially #####

rev_trans12=scaler.inverse_transform(df_pred12)
```

```
###### Extraction of Bi-LSTM prediction vector and real values vector #####
df_final12=df_input[-prediction21.shape[0]:]
df011=df_final12['T2M']
rev022=rev_trans12[:,0]


##### Ploting of Bi-LSTM prediction vs real observation values of T2M #####

dates = pd.date_range('08/21/2020', periods=498)
Forecastb1=pd.DataFrame({'Real values of T2M':df011, 'Bi-LSTM Predicted Values of T2M':
Forecastb1['date']=dates
Forb1=Forecastb1.set_index('date')
Forb1.plot.line(figsize=(15, 4),color=['peru','forestgreen'])
plt.ylabel("T2M",fontsize=18)
plt.xlabel("Date",fontsize=18)
plt.title('Bi-LSTM for Temperature prediction')
plt.legend(fontsize=15)
plt.show()



                 ##############################
                 ####                      ####
                 ####   Wind speed (WS2M)  ####
                 ####      prediction      ####
                 ##############################


#####################################################
###### LSTM for wind speed (WS2M) prediction   ######
#####################################################

########Datapreprocessing For Wind speed ##########

######Creating another data by changing
the column and sorting the index######

df_input= df[['WS2M','T2M', 'QV2M', 'RH2M', 'PREC', 'PS','SWRD']].sort_index()
scaler=MinMaxScaler()
data_scaled=scaler.fit_transform(df_input)
# define X and Y
window_length= 14
b_size= 32
```

```
target=data_scaled[:,0]
features=data_scaled
X_train,  y_train = features[:1636,:], target[:1636]
X_val,y_val= features[1636:2045,:], target[1636:2045]
X_test,y_test= features[2045:,:], target[2045:]
# Train and test generator
features_num=7
train_generator=TimeseriesGenerator(X_train,y_train
,length=window_length,sampling_rate=1,batch_size=b_size)

val_generator= TimeseriesGenerator(X_val,y_val
,length=window_length
,sampling_rate=1,batch_size=b_size)

test_generator= TimeseriesGenerator(X_test,y_test
,length=window_length,sampling_rate=1,batch_size=b_size)


###########Model Construction###########
model=Sequential()
model.add(LSTM(128, activation='relu',
input_shape=(window_length, features_num), return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(1))

model.summary()

########Early stopping: Model stops if
the validation loss does not improve after 3
more iteration the algorithm stop############



early_stopping= tf.keras.callbacks.EarlyStopping(monitor='val_loss'
,patience=60,mode='min')
model.compile(loss=tf.losses.MeanSquaredError(),
optimizer='adam'
,metrics=[tf.metrics.MeanAbsoluteError()
,tf.metrics.RootMeanSquaredError()])

history=model.fit(train_generator,epochs=90,shuffle=False
,validation_data=val_generator,callbacks=[early_stopping])
###### Plot of Training error and validation error (LEARNING CURVE) ######

pd.DataFrame(history.history)[['loss','val_loss']].plot(figsize=(8,5))
```

```
plt.ylabel("Cost function")
plt.xlabel("Epoch")
plt.show()
pd.DataFrame(history.history)[['mean_absolute_error'
,'val_mean_absolute_error']].plot(figsize=(8,5))
plt.ylabel("Error")
plt.xlabel("Epoch")
plt.show()



##### Evaluation of the model #####
model.evaluate(test_generator,verbose=0)

##### Predictions construction #####
predictions=model.predict(test_generator)
# print(predictions.shape), print(y_test.shape),print(New_X_test.shape)
# Consider the values of the X_test by ignoring the first 14th
New_X_test=X_test[:,1:][window_length:]

##### Datapredicted ####
df_pred=pd.concat([pd.DataFrame(predictions)
,pd.DataFrame(New_X_test)],axis=1)

#### return the inverse transform of the
MinMaxscaler created above to give the
real value that we initially had ####

rev_trans=scaler.inverse_transform(df_pred.values)

####### Prediction vector and real values vectors construction ###########


df_final=df_input[-predictions.shape[0]:]
df1=df_final['WS2M']
rev2=rev_trans[:,0]


####### Performance metrics for LSTM #######
mse = mean_squared_error(df1, rev2)
r2 = r2_score(df1, rev2)
mae = mean_absolute_error(df1, rev2)
# mae_datascaled = mean_absolute_error(X_test[-rev2.shape[0]:,0], predictions[:,0])
mape = mean_absolute_percentage_error(df1, rev2)
print("mse=" + str(round(mse,3)))
print("r2=" + str(round(r2,3)))
print("mae=" + str(round(mae,3)))
# print("mae_datascaled=" + str(round(mae_scaled,2)))
print("mape=" + str(round(mape,3)))

######Plot of LSTM Predicted values vs real values of the variable WS2M ########

dates = pd.date_range('08/21/2020', periods=498)
Forecastb1=pd.DataFrame({'Real values of WS2M':df1, 'Predicted Values of WS2M':rev2})
Forecastb1['date']=dates
Forb1=Forecastb1.set_index('date')
Forb1.plot.line(figsize=(15, 4),color=['peru','forestgreen'])
plt.title("LSTM for wind speed prediction")
plt.ylabel("WS2M",fontsize=18)
plt.xlabel("Date",fontsize=18)
plt.legend(fontsize=15)
plt.savefig('LSTM_TMP_PREDx.png')
files.download("LSTM_TMP_PREDx.png")
plt.show()

#####################################################
###### Bi-LSTM for wind speed prediction (WS2M)######

#####################################################

model2= keras.Sequential()

model2.add(
    layers.Bidirectional(layers.LSTM(94, return_sequences=False)
    , input_shape=(window_length, features_num))
)
model2.add(layers.Dense(1))
model2.summary()
#######Early stopping: Model stops if the validation loss does
not improve after 50 more iterations
the algorithm stop######

early_stopping1=tf.keras.callbacks.
EarlyStopping(monitor='val_loss',patience=50,mode='min')
model2.compile(loss=
tf.losses.MeanSquaredError()
,optimizer='adam'
,metrics=[tf.metrics.MeanAbsoluteError()
,tf.metrics.RootMeanSquaredError()])
history2=model2.fit(train_generator,epochs=150
,shuffle=False, validation_data=val_generator,callbacks=[early_stopping1])


###### Predictions ######

prediction2=model2.predict(test_generator)

#######Consider the values of X_test by ignoring the first 14th######
New_X_test2=X_test[:,1:][window_length:]

#######Data predicted######

df_pred12=pd.concat([pd.DataFrame(prediction2)
,pd.DataFrame(New_X_test2)],axis=1)

#######return the inverse transform of
the MinMaxscaler created above
to give the real value that we had initially######
rev_trans12=scaler.inverse_transform(df_pred12)


#####Dysplay the prediction vector
and the real values vectors######

df_final12=df_input[-prediction2.shape[0]:]
df011=df_final12['WS2M']
rev022=rev_trans12[:,0]

######Plot of Bi-LSTM prediction
and real values of WS2M######
dates = pd.date_range('08/21/2020', periods=498)
Forecastb1=pd.DataFrame({'Real values of WS2M':df011
,'Bi-LSTM Predicted Values of  WS2M':rev022})
Forecastb1['date']=dates
Forb1=Forecastb1.set_index('date')
Forb1.plot.line(figsize=(15, 4),color=['peru','forestgreen'])
plt.ylabel("WS2M",fontsize=18)
plt.xlabel("Date",fontsize=18)
plt.title("Bi-LSTM prediction for WS2M ")
plt.legend(fontsize=15)
plt.show()

###### Performance metrics for Bi-LSTM ######

mse1 = mean_squared_error(df011, rev022)
```

```
r2_1 = r2_score(df011, rev022)
mae1 = mean_absolute_error(df011, rev022)
# mae1_scaled1 = mean_absolute_error(X_test[-df_pred.shape[0]:,0], prediction[:,0])
mape1 = mean_absolute_percentage_error(df011, rev022)
print("mse=",mse1)
print("r2=",r2_1)
print("mae=",mae1)
# print("mae_scaled=" + str(round(mae1_scaled1,2)))
print("mape=",mape1)


                      ##############################
                      ####                      ####
                      #### Precipitation (PREC) ####
                      ####      prediction      ####
                      ##############################


df_input= df[['PREC','T2M', 'WS2M', 'QV2M', 'RH2M', 'PS','SWRD']].sort_index()
scaler=MinMaxScaler()
data_scaled=scaler.fit_transform(df_input)

#####################################################
###### LSTM for precipitation (PREC) prediction #####

#####################################################

###### Datapreprocessing for precipitation#####

# define X and Y
target=data_scaled[:,0]
features= data_scaled
features_num=7
X_train,  y_train = features[:1636,:], target[:1636]
X_val,y_val= features[1636:2045,:], target[1636:2045]
X_test,y_test= features[2045:,:], target[2045:]
# Train and test generator
window_length= 14
b_size= 32
features_num=7
train_generator= TimeseriesGenerator(X_train,y_train
,length=window_length,sampling_rate=1,batch_size=b_size)
val_generator= TimeseriesGenerator(X_val,y_val
,length=window_length,sampling_rate=1,batch_size=b_size)
test_generator= TimeseriesGenerator(X_test,y_test
,length=window_length,sampling_rate=1,batch_size=b_size)

################## Model construction ##################
model=Sequential()
model.add(LSTM(128, activation='relu'
, input_shape=(window_length, features_num), return_sequences=False))
# model.add(LSTM(24
, activation='relu', return_sequences=True))
# model.add(LSTM(32, activation='relu'
, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(1))

model.summary()

########  Early stopping: Model stops if the
validation loss does not improve after
45 more iterations the algorithm stop ##########


early_stopping= tf.keras.callbacks.EarlyStopping(
monitor='val_loss',patience=45,mode='min')
model.compile(loss=tf.losses.MeanSquaredError()
,optimizer='adam'
,metrics=[tf.metrics.MeanAbsoluteError()
,tf.metrics.RootMeanSquaredError()])
history=model.fit(train_generator,epochs=150,shuffle=False, validation_data=val_generat

########  Learning curves ########

pd.DataFrame(history.history)[['loss','val_loss']].plot(figsize=(8,5))
plt.ylabel("Cost function")
plt.xlabel("Epoch")
plt.show()
pd.DataFrame(history.history)[['mean_absolute_error'
,'val_mean_absolute_error']].plot(figsize=(8,5))
plt.ylabel("Error")
plt.xlabel("Epoch")
plt.show()

######### Evaluation of the model ########
model.evaluate(test_generator,verbose=0)

# Predictions
predictionsp=model.predict(test_generator)
```

```
# print(predictions.shape), print(y_test.shape),print(New_X_test.shape)
# Consider the values of X_test by ignoring the first 14th
New_X_testp=X_test[:,1:][window_length:]

# Datapredicted
df_predp=pd.concat([pd.DataFrame(predictionsp)
,pd.DataFrame(New_X_testp)],axis=1)
#########  return the inverse transform of the
MinMaxscaler created above to give the
real value that we had initially ########
rev_transp=scaler.inverse_transform(df_predp)

#########  LSTM prediction and real values of PREC ########

df_finalp=df_input[-predictionsp.shape[0]:]
df1p=df_finalp['PREC']
rev2p=rev_transp[:,0]


######### Performance metrics ############

msep = mean_squared_error(df1p, rev2p)
r2p = r2_score(df1p, rev2p)
maep = mean_absolute_error(df1p, rev2p)
mapep = mean_absolute_percentage_error(df1p, rev2p)
print("mse=", msep)
print("r2=",r2p)
print("mae=",maep)
print("mape=" ,mapep)

###### Comparison  Plot ######
dates = pd.date_range('08/21/2020', periods=498)
Forecastb1=pd.DataFrame({'Real values of the variable Precipitation':df1p, 'LSTM Predic
Forecastb1['date']=dates
Forb1=Forecastb1.set_index('date')
Forb1.plot.line(figsize=(15, 4),color=['peru','forestgreen'])
plt.ylabel("PREC",fontsize=18)
plt.xlabel("Date",fontsize=18)
plt.title("LSTM for precipitation prediction ")
plt.legend(fontsize=15)
plt.show()



#####################################################
```

```
######Bi-LSTM for precipitation (PREC) prediction ###

#####################################################

model2= keras.Sequential()

model2.add(
    layers.Bidirectional(layers.LSTM(194, return_sequences=False), input_shape=(window_
)
model2.add(layers.Dense(1))
model2.summary()
####### Early stopping: Model stops if the validation
loss does not improve after 50 more iterations
the algorithm stop ######

early_stopping1= tf.keras.callbacks.
EarlyStopping(monitor='val_loss',patience=50,mode='min')
model2.compile(loss=tf.losses.MeanSquaredError()
,optimizer='adam',metrics=[tf.metrics.MeanAbsoluteError()
,tf.metrics.RootMeanSquaredError()])
history2=model2.fit(train_generator,epochs=95,shuffle=False, validation_data=val_genera



###### Learning curves ######


pd.DataFrame(history2.history)[['loss'
,'val_loss']].plot(figsize=(8,5))
plt.ylabel("Cost function")
plt.xlabel("Epoch")
plt.savefig('Bi-LSTM_PREC_LOSS.png')
files.download("Bi-LSTM_PREC_LOSS.png")
plt.show()
pd.DataFrame(history2.history)[['mean_absolute_error'
,'val_mean_absolute_error']].plot(figsize=(8,5))
plt.ylabel("error")
plt.xlabel("Epoch")
plt.savefig('Bi-LSTM_PREC_ERROR.png')
files.download("Bi-LSTM_PREC_ERROR.png")
plt.show()


# Predictions
prediction2=model2.predict(test_generator)
```

```
# Consider the values of X_test by ignoring the first 14th
New_X_test2=X_test[:,1:][window_length:]

# Datapredicted
df_pred12=pd.concat([pd.DataFrame(prediction2),pd.DataFrame(New_X_test2)],axis=1)
###### return the inverse transform of the MinMaxscaler created above to give the real
rev_trans12=scaler.inverse_transform(df_pred12)



##### Extraction of the Bi-LSTM prediction and real values of PREC  #####

df_final12=df_input[-prediction2.shape[0]:]
df011=df_final12['PREC']
rev022=rev_trans12[:,0]


#####  Comparison Plot between Bi-LSTM prediction and real observation values of PREC #

dates = pd.date_range('08/21/2020', periods=498)
Forecastb1=pd.DataFrame({'Real values of the variable Precipitation':df011, 'Bi-LSTM Pr
Forecastb1['date']=dates
Forb1=Forecastb1.set_index('date')
Forb1.plot.line(figsize=(15, 4),color=['peru','forestgreen'])
plt.ylabel("PREC",fontsize=18)
plt.xlabel("Date",fontsize=18)
plt.title("Bi-LSTM for precipitation prediction ")
plt.legend(fontsize=15)
plt.show()
```