School of Mathematics and Statistics

Weather forecasting using recurrent neural networks

Student: Enrico Porcelli

Project supervisor: Vladislav Vyshemirsky

Contents

1 Introduction

The aim of this project is to study the effectiveness of a recurrent neural network in the forecasting of time series by comparing its predictions to the ones of simpler models such as an ARIMA model and a baseline model. The dataset that will be used is a weather time series dataset recorded by the Max Planck Institute for Biogeochemistry located in Jena, Germany. The study will be performed on a smaller portion of the data collected, in particular the section that goes from the start of 2009 to the end of 2016. The uncut data collection started on November 24th 2003 and is still going on nowadays. The variables collected are 14 and, except for the one describing the time of collection, they are all weather variables. All observations included in the following study have been collected every 10 minutes with the last observation being collected at 00:00:00 of 01/01/2017. The following is a list of the variables collected, along with a short explanation of what they measure:

- **Date.Time**: date and hour of collection;
- **p..mbar.**: air pressure in mbar;
- **T..degC.**: air temperature in °C;
- **Tpot..K.**: potential temperature, measured in °K. The potential temperature of a parcel of fluid is the temperature that this parcel would reach if adiabatically brought to the standard pression of 1000 mbar;
- **Tdew..degC.**: dew point temperature. It is measured in °C and it is the temperature to which air must be cooled to reach saturation. The higher the dew point, the more moisture there is in the air;
- **rh. . . .**: relative humidity, measured in percentage. Relative humidity is defined as the ratio of the current absolute humidity to the highest possible absolute humidity given the air temperature. A 100% relative humidity means that the air is totally saturated with moist;
- **VPmax..mbar.**: saturation vapor pressure, measured in mbar. It is defined as the pressure at which the air will be saturated given the current temperature;
- **VPact..mbar.**: vapor pressure, measured in mbar. It is defined as the pressure exerted by a vapor in thermodynamic equilibrium with its condensed phases (solid or liquid) at a given temperature in a closed system;
- **VPdef..mbar.**: vapor pressure deficit, measured in mbar. It can be computed as the difference between VPmax..mbar. and VPact..mbar.;
- **sh..g.kg.**: specific humidity, measured in g/kg. Specific humidity is the ratio of the mass of water vapor to the total mass of the air parcel;
- **H2OC..mmol.mol.**: water vapor concentration, measured in mmol/mol;
- **rho..g.m..3**: air density, measured in g/m^3. It is the mass per unit volume of Earth's atmosphere;
- **wv..m.s.**: wind velocity, measured in m/s;
- **max.wv..m.s.**: maximum wind velocity, measured in m/S. It is the maximum wind speed collected during the collection of the variable wv..m.s.;
- **wd..deg.**: wind direction, measured in degrees;

The dependent variable for this project is $T..degC.$ and so that is what the forecasts will try to predict. The type of regression will be either multivariate or univariate, depending on the tool used for computing the predictions. The recurrent neural network is able to deal with multidimensional inputs, while the ARIMA model and the baseline model will be fitted using only the dependent variable. Here is a brief description of the three tools that will be used for forecasting in this project:

- **Baseline model**: this model predicts the next time step by simply copying the value at the current timestep. This means that if the current temperature is 8.3°C, the prediction that this model will make for the temperature that will be measured in one or more hours is 8.3°C;
- **ARIMA model**: this kind of model has memory along the time line and so it makes sense to adopt it for modelling a time series. Moreover, since the observations are weather related, they will have a cyclical component related to the day/night cycle and one related to the seasons cycle. Both of these seasonalities can be captured using an ARIMA model;
- **Recurrent neural network**: a problem with classic artificial neural networks is that they do not have any kind of memory, the prediction for time step t is not influenced by the observation collected at time step t-1. This is the reason behind the decision of using a recurrent neural network in this project: this particular type of neural network takes into consideration the data from the previous time steps by utilising as input both the observation collected at time t and the output produced by the hidden layer at time step t-1. This form of "memory" is yet not optimal, but it can be improved with more complex architectures such as the long-short term memory, or LSTM, which is the one that has been used in this project.

2 Data manipulation and exploratory analysis

Since the aim is to make hourly predictions, it will be used a subset of the original dataset containing one observation per hour.
A first analysis of the dataset revealed that it does not contain missing values and, by looking at the description of the variables, it can be noticed that the variable $VPdef..mbar$ is just the difference between $VPmax..mbar$ and $VPact..mbar$, this means that its information is redundant and the corresponding column can be erased without losing any information. After doing these first steps, the first few rows of the dataset will look like this:

Table 1: First five lines of the dataset used in the project

Date.Time	p..mbar.	T..degC.	Tpot..K.	Tdew..degC.	rh. . . .
01.01.2009 01:00:00	996.50	-8.05	265.38	-8.78	94.4
01.01.2009 02:00:00	996.62	-8.88	264.54	-9.77	93.2
01.01.2009 03:00:00	996.84	-8.81	264.59	-9.66	93.5
01.01.2009 04:00:00	996.99	-9.05	264.34	-10.02	92.6
01.01.2009 05:00:00	997.46	-9.63	263.72	-10.65	92.2

It can be noticed that the *Date.Time* variable is useless at the moment: it cannot be treated as a factor because it would have too many levels and it can not be ignored either. The reason is that since the dataset is a collection of time series data, the seasonalities will have a big impact on the results and the *Date.Time* variable is the only one containing information about the time of the day and the time of the year in which each observation has been collected. To give this variable a meaning, each of the *Date.Time* observations will be transformed in the equivalent amount of seconds passed since 01-01-2009 01:00:00, i.e. the time at which the first observation has been collected. The new variable *seconds* will then have these values:

```
0 3600 7200 10800 14400 18000 21600 25200 ...
```

Now that there is a variable to describe the time feature of each observation, it must be transformed into something that is able to capture seasonalities. This transformation is needed because the time series of the response variable clearly shows seasonality.

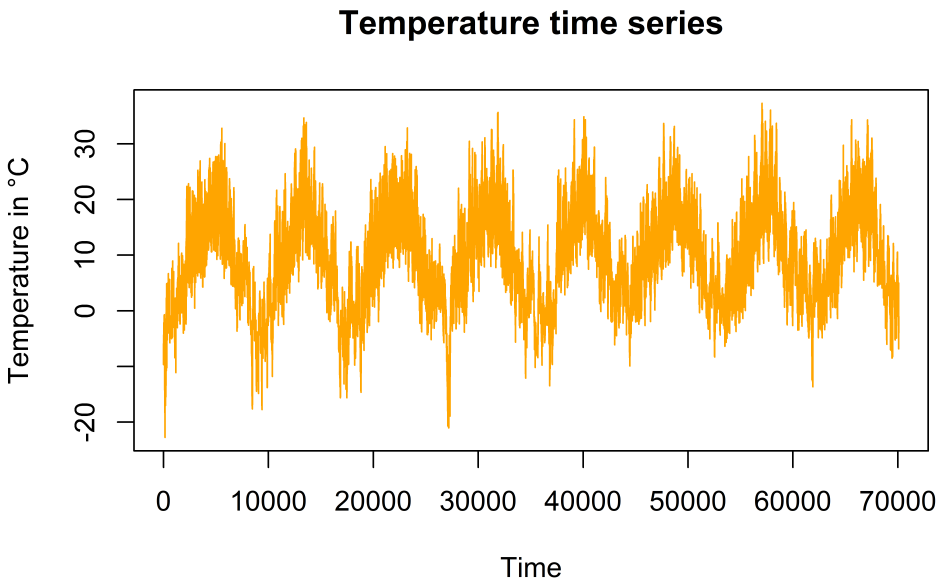

Figure 1: *Graph of the time series describing the temperature in celsius degrees, which is the dependent variable for this project.*

To explain why the current *seconds* variable is not optimal in this situation, an example will be considered.
Let's take the first observation (collection time: 01-01-2009 01:00:00) and the one collected at the same time of the day but five months later (collection time: 01-05-2009 01:00:00). Their difference in the time component is 10364400 seconds and this can be useful for describing a linear trend between time and temperature in these five months. The problem, however, is that this value does not highlight an important feature which is common for both observations

and that is crucial in time series analysis: these two observations are both located in the same spot in the day/night cycle and hence they will have common characteristics (e.g. having lower temperature than the other observations collected in the same days). Similarly, the same happens when comparing the first observation with the one collected after exactly one year: their position in the yearly cycle is not considered at all if only the current variable *seconds* is considered. To solve this problem the just created *seconds* variable has been expressed in a two dimensional vector containing its sine and cosine with respect to the position of the observation in the daily cycle. This will give to every observation in a given day a unique combination of a sine and cosine value. In this way each observation in a day is uniquely identified (as it was with just the variable *seconds*) but with the advantage that now observations collected at similar times but in different days will have a common characteristic. The same procedure has to be applied in order to account for the annual cycle, what changes is just the period of the sine and cosine transformations. With this whole procedure, each observation of the variable *seconds* will be transformed into a four dimensional vector with the following entries:

For the day/night cycle:

$$sin.time.day = sin(\frac{seconds * 2\pi}{24 * 60 * 60})$$

$$cos.time.day = cos(\frac{seconds * 2\pi}{24 * 60 * 60})$$

For the yearly cycle:

$$sin.time.year = sin(\frac{seconds * 2\pi}{365 * 24 * 60 * 60})$$

$$cos.time.year = cos(\frac{seconds * 2\pi}{365 * 24 * 60 * 60})$$

Next step in the data manipulation process is to identify outliers and for this issue a box plot (Tukey, 1977) will be used. The only box plots that revealed the presence of outliers are the ones for the variables measuring air pressure, wind speed and maximum wind speed.

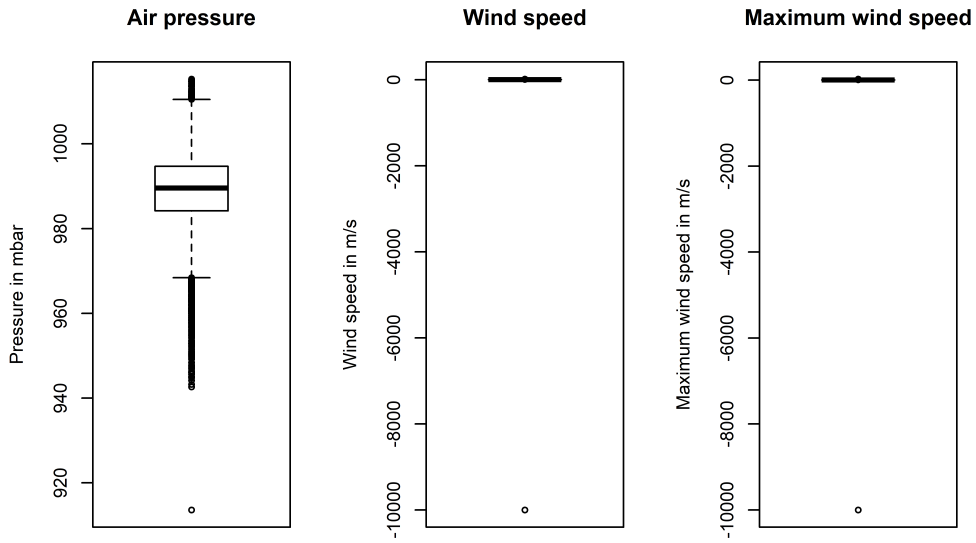

Figure 2: *Boxplots of the three variables in the dataset that showed the presence of outliers*

To solve the outliers issue in the wind variables, the following technique has been used when spotting a suspicious observation: wind speed has been set to be equal to zero and maximum wind speed has been set to be equal to the wind speed. This last substitution has been adopted because of a property of the mean: the mean always lies between the range of the data. It follows that setting the maximum equal 0, as has been done for the wind speed, may result in some observations having an average wind speed higher than the maximum wind speed, which is impossibile.
If for these two variables the typo is clear (a wind speed of -9999.0 is definitely wrong), for air pressure the outlier must be analysed more carefully. Let's look at the time series of the air pressure around the suspicious observation.

```
977.99 977.57 913.6 974.33 977.79 ...
```

This must be an outlier since such a significant drop in air pressure is possible only during hurricanes and no hurricane has been recorded between 2009 and 2016 in Jena, the location where the data have been collected. To have a valid value for the air pressure, the observation has been subsituted by the mean of the previous and next observations.

The next transformation regards wind direction and, as for the $Date.Time$ variable, a higher dimensional vector will be created. It will contain the *wd..deg.* transformed by a sine and cosine transformation. This must be done because a wind that blows at a 359° angle should be considered similar to one that blows at 1° angle and one way to solve the problem is to express the angle in term of its sine and cosine.

Now the original *wd..deg.* column can be eliminated, the *seconds* column has been kept to capture possible linear trends in time.

Now it is time to check for correlation between the variables in the dataset, in order to reduce redundant information and avoid multicollinearity issues. This will be done by analysing the Pearson correlation coefficient (Pearson, 1896).

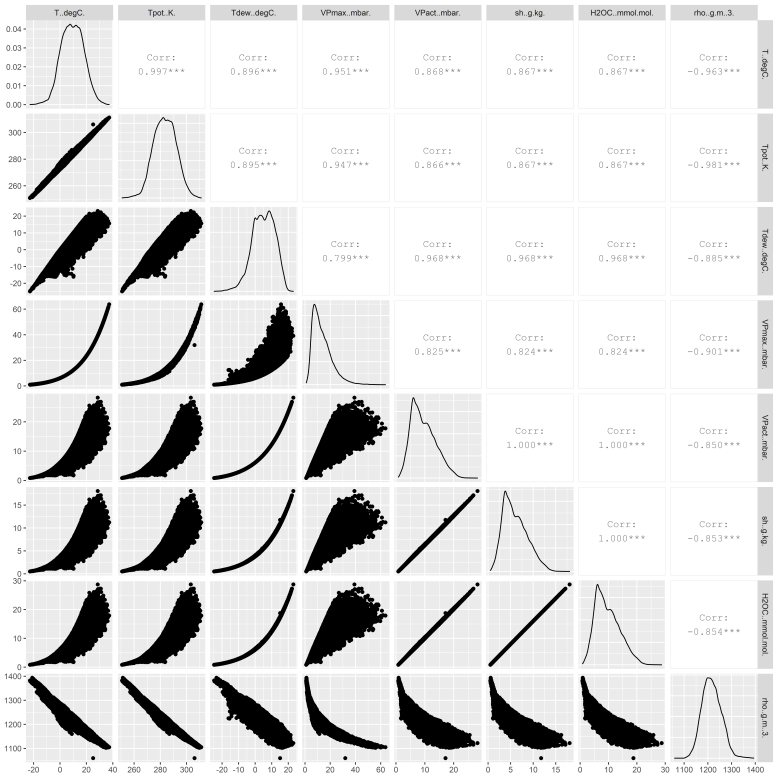

Figure 3: *Variables against variables plot for all the variables present in the dataset. The value symmetric to each plot with respect to the main diagonal is the Pearson's correlation coefficient for the two variables in the plot.*

As the graph shows, the variable $VPact..mbar$ is linearly correlated with $sh..g.kg$ and with $H2OC..mmol.mol$, hence the last two can be erased while keeping only $VPact..mbar$ (which is the one with the highest correlation with the response variable), otherwise multicollinearity may arise. The last group of variables for which a decision has to be made is composed of $TDew..DegC$, $rho..g.m..3$, $Tpot.K$ and $VPMax..mbar$. They are all highly correlated with each other and with the response variable but, again, $Tpot.K$ is the most correlated with $T..degC.$ and so it is the one that has been kept in the dataset.

Now the only thing left to do before starting modelling is to standardize the data. This is necessary when using machine learning because the algorithms in this field work better with less spread values. Right now some variables have too much variability.

After doing that the dataset is ready to be used both in a recurrent neural network and in simpler models such as the baseline and ARIMA model.

3 Methodology

3.1 Neural networks

A neural network (McCulloch & Pitts, 1943) is a series of algorithms that aims at recognizing underlying relationships between input and output variables. In order to achieve this, a series of weighted sums and non linear transformations are applied to the input values to obtain a final prediction, which will be a function of all the weights used in the different weighted sums. After that, by comparing the final prediction to the ground truth, the weights are adjusted according to the partial derivative of the cost function with respect to each singular weight. Neural networks owe the adjective "neural" to the fact that they are structured in order to mimic what human neurons do, this is the reason why each single computational unit in neural networks is called neuron. This project will focus on a particular kind of neural networks which is called recurrent neural network, RNN for short.

3.2 Recurrent neural network architecture

3.2.1 Standard recurrent neural network

A recurrent neural network (Rumelhart et al., 1985), or RNN for short, is a particular kind of neural network which possesses some interesting characteristics that made it gain a lot of popularity in the last decade, mainly in natural language tasks and time series forecasting. Compared with a standard artificial neural network, the upgrade that this method brings in forecasting tasks is the fact that the output at any given time step is involved in the computation of the output at next time step. This characteristic gives the RNN the ability to learn temporal relationships between observations.

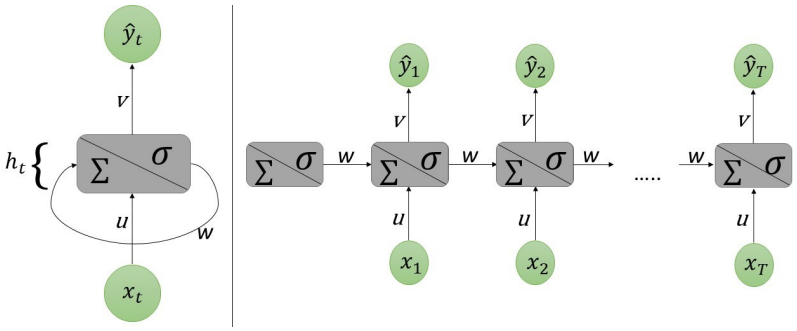

Figure 4: *Scheme of the simplest Recurrent Neural Network. On the left it is shown in its iterative representation, on the right what is shown is its unrolling through time.*

The structure seen in the picture is divided in three layers: an *input layer*, a *hidden layer* and an *output layer*. Each layer is composed, in the simplest possible RNN, of a single neuron, which is a computational unit that performs some transformations on the data received as input. The neuron in the input layer performs the identity transformation, i.e. it simply passes the input value to the neuron in the hidden layer without applying any transformation. When the hidden layer receives the value, it uses it to perform a weighted sum and a non linear transformation. The result of these transformations performed at time step t in the hidden layer is called the *hidden state at time t*. After the computation, the hidden state is passed through the network both to the output layer and to the hidden layer at next time step, where it will be treated as an input for the weighted sum. In the output layer a weighted sum and a linear or non linear transformation are applied to the input received to produce a prediction. The reason behind the word "recurrent" in the name RNN is the procedure followed by the hidden layer, i.e. sending the hidden state at time t to the hidden layer at time $t+1$. Theoretically speaking, this means that the last output at time $t = T$ depends on all the hidden states at time $t = T-1, T-2, ..., 1$ and, consequently, on all the corresponding inputs. This recurrent structure means that the neuron placed in the hidden layer of a RNN will receive two inputs at any time step: one is a new input value, the other is the output given by the same neuron at previous time step. This is what gives the RNN a sort of "memory" that can help them to make more precise time series forecasts than a simple artificial neural network. To explain it more clearly it is necessary to define the notation that will be used for explaining the simplest form of RNN: the one with a scalar input, a scalar output and a single neuron in the hidden layer. To ease the notation it will now be specified, and never again repeated, that t ranges from 1 to T and that the weights are randomly initialized at first

- x_t: the scalar input at time step t;
- h_t: the state of the hidden layer at time t;
- $\hat{y}_t$: output at time step t;
- z_t: value of the weighted sum at the hidden state at time step t, t = 1,. . . ,T
- u: the weight that connects the input to the hidden layer;
- w: the weight that connects h_{t-1} (previous state of the hidden layer) with the recurrent hidden layer at time t;
- v: the weight that connects h_t with the output $\hat{y}_t$.

The quantities that are not explicit are z_t, h_t and $\hat{y}_t$. These are defined as follows:

$$z_t = ux_t + wh_{t-1}$$
$$h_t = \sigma(z_t)$$
$$\hat{y}_t = g(vh_t)$$

So the "journey" performed by every input value at any time step t is the following:

$$x_t \ \rightarrow \ z_t = x_t u + wh_{t-1} \ \rightarrow \ h_t = \sigma(z_t) \ \rightarrow \ \hat{y} = g(vh_t)$$

where h_0 is initialised to be equal to 0. The function g is usually a linear function in simpler models, while it is a nonlinear one if more complex structures are used. The function $\sigma(z_t)$ is called *activation function* and is a non linear function that transforms the weighted sum of the inputs received by the hidden layer at time t. If every function applied to x_t were linear, then the whole neural network would correspond to a single linear transformation and it would not be able to identify complex relationships between data. This advantage of non linear activation functions will be easier to understand when dealing with more complex RNN structures, where multiple non linear transformations are applied one after another. The most commonly used activation functions in RNN are sigmoid functions like the logistic function and the hyperbolic tangent. Other popular options are functions like the Rectified Linear Unit function (ReLU). They are defined as follows:

- *Logistic function* :

$$S(x) = \frac{e^x}{e^x + 1}$$

  This function transforms every result of the weighted sum into a value between 0 and 1. The disadvantage of this function is that it tends to saturate, meaning that high values of the sum will be very close to 1 and low values will be very close to 0. The only area in which significant changes in x correspond to significant changes in S(x) is the one near zero, where the function is more steep. Moreover, it can cause a problem called "vanishing gradient", which will be explained later.

- *Hyperbolic tangent* :

$$tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

  Similar to the logistic function but can take values in the range (-1,1). The disadvantages are the same as the just mentioned function.

- *Rectified linear unit (ReLU)* :

$$R(x) = max(0, x)$$

  If the result of the weighted sum is negative, then the ReLU function (Nair & Hinton, 2010) takes the value 0, otherwise it takes the value of the weighted sum. In this way a neuron may be "switched on/off", meaning that it has recognised a precise pattern or not. It is frequently used in natural language tasks or in sentimental analysis as a last transformation in the output layer.

Once the input has been transformed, this process is iterated for every time step and a final result is obtained. Depending on the task we have to solve, a different kind of RNN with different activation functions may be used.

3.2.2 Vector hidden Layer RNN

This just mentioned RNN architecture is easy to understand but it struggles when trying to track complex data structures. Adding neurons to the hidden layer gives to the algorithm more flexibility, making it able to predict more challenging patterns in the data. Let's see how the dimensionalities of the parameters of the network change with this new architecture.

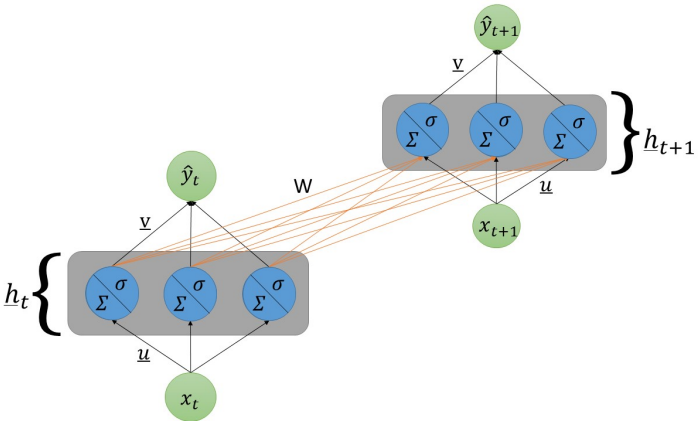

Figure 5: *Graphic representation of a Recurrent Neural Network having several neurons in the hidden layer. Every line connecting two neurons represents a particular weight in the netwwork. In this figure it is shown the unrolling through time between time steps t and t+1*

- x_t: the input, is still a scalar one;
- $\underline{h}_t$: it is a vector belonging to $\mathbb{R}^d$. This vector has as d-th entry the state of the d-th neuron of the hidden layer at time t;
- $\underline{u}$: the weight vector belonging to $\mathbb{R}^d$. The entries of this vector are $[u_1, u_2, ..., u_d]$ which are the weights connecting the scalar input to each single neuron of the hidden layer;
- W: the weight matrix belonging to $\mathbb{R}^{d\times d}$. Its element are indexed as $[w_{jk}]$ where the subscripts j and k indicate that that single entry is the weight that connects the k-th hidden neuron at time step $t-1$ with the j-th one at time step t. This notation may seem backward at first but it will simplify the computations, allowing to avoid transposing the weight matrix;
- $\underline{v}$: the weight vector belonging to $\mathbb{R}^d$. The entries of this vector are $[v_1, v_2, ..., v_d]$ which are the weights connecting the scalar input to each single neuron of the hidden layer.

3.2.3 Vector input/output and hidden layer

The initial restriction that obligated the input and the output to be scalar was too restrictive. In real life, input and outputs are often vectors and to process them, a RNN should have more neurons in the input/output layer. In particular it should have in the input/output layer as many neurons as it is the dimensionality of the input/output vector. This structure is one of the most commonly used.

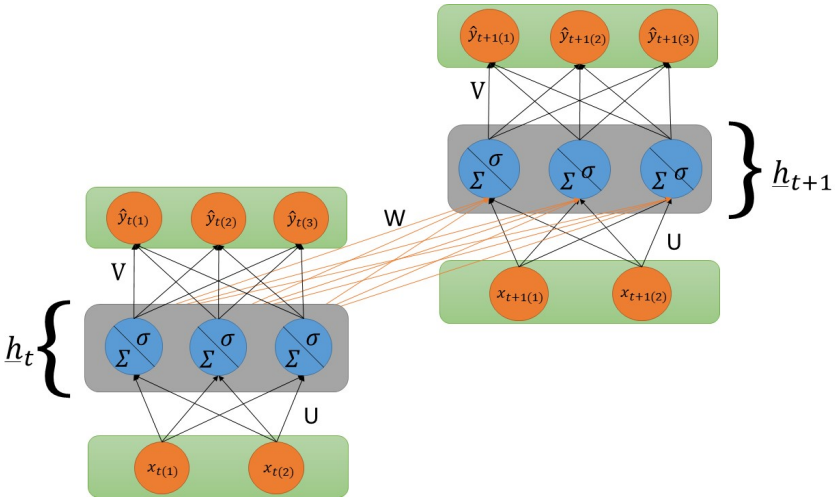

Figure 6: *Graphic representation of a Recurrent Neural Network having several neurons in the hidden layer and having multidimesnional inputs and outputs. Every line connecting two neurons represents a particular weight in the netwwork. In this figure it is shown the unrolling through time between time steps t and t+1*

Now the notation should be updated as follows:

- $\underline{x}_t$: the input vector of dimension $\mathbb{R}^k$;
- $\underline{h}_t$: it is a vector belonging to $\mathbb{R}^d$. This vector has as entries the states of the single d neurons of the hidden layer;
- $\underline{\hat{y}}_t$: the output vector belonging to $\mathbb{R}^p$;
- U: the weight matrix belonging to $\mathbb{R}^{d\times k}$. The entries $[u_{ij}]$ of the matrix are the weights connecting the j-th neuron of the input layer with the i-th neuron of the hidden layer.
- W: same as in the previous case;
- V: the weight matrix belonging to $\mathbb{R}^{p\times d}$. The entries $[v_{ij}]$ of the matrix are the weights connecting the j-th neuron of the hidden layer with the i-th neuron of the output layer.

This structure will be used in this project with fixed parameters $k = 16$, $d = 32$ and $p = 1$.

3.2.4 Vector input, vector hidden Layer and a deep RNN

In the previous two RNN structures the input data have been transformed through a non linear activation function to give the algorithm more flexibility. One non linear transformation, however, is often not enough to mimic the complex processes of the real world and hence a structure with more than one hidden layer (which translates in more than one non linear transformation) may be necessary. This type of RNN is called deep RNN and is built by stacking several hidden layers that may be composed of multiple neurons. These kinds

of RNN are rare because having multiple recurrent layers would be very computationally demanding, that's why sometimes it is preferrred to have just one or two hidden recurrent layers and then a series of hidden layers that are not recurrent (and hence less computationally demanding). An example of deep RNN will be shown but it will not be used in this project.

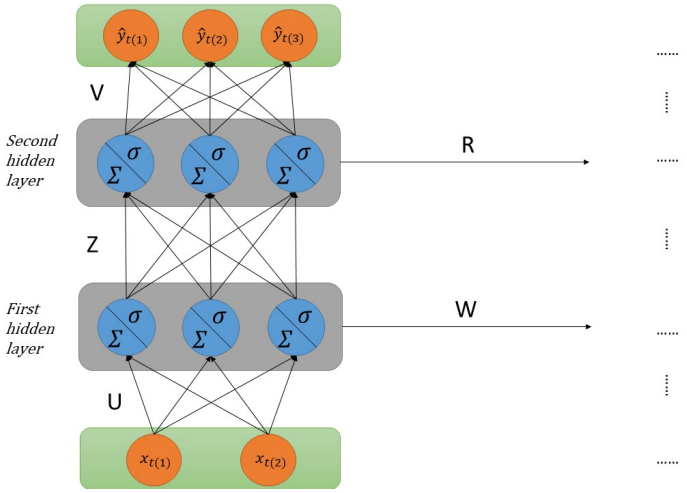

Figure 7: *Graphic representation of a deep Recurrent Neural Network having two recurrent hidden layers.*

3.3 Training a recurrent neural network

Recurrent neural networks can manage a sequence of inputs and they can output a single value as well as a sequence of values. The so called "training" of the network is the phase in which the weights of the network are modified in order to achieve the best possible prediction. The training phase is a supervised learning algorithm, meaning that the values that the RNN is trying to predict are already known. What is crucial to consider is that the RNN is trained for a specific purpose, e.g., for the prediction of a value 3h into the future or for the prediction of a sequence of 15 hours of observations, and so the weights are optimized for that purpose only.

3.3.1 Backpropagation through time

Backpropagation through time (Mozer, 1995) is an algorithm that aims at finding a local or global minimum in the cost (also called loss) function by changing the weights of the model. The mathematical tools used are the partial derivatives and the chain rule. To ease the notation, weights will be indicated with upper case letters U, V and W, while input, outputs, hidden states and true values will respectively be indicated as x_t, $\hat{y}_t$, h_t and y_t. This avoids underlining every vector, which may make the equations hard to read.

To explain how backpropagation through time works let's see a specific example in which the aim of the neural network is to predict a single value one hour apart from the last observation given in the input window. Let's choose the cost (or loss) function to be:

$$C_t = \frac{1}{2}(\hat{y}_t - y_t)^2$$

Which is a common cost function called Mean Squared Error or MSE. In the general formula of the MSE, instead of $\frac{1}{2}$, an unspecified constant c is used. In machine learning this constant is almost always chosen to be $\frac{1}{2}$ because this choice simplifies the computation of the derivatives. For the sake of simplicity it is assumed that g, which is the linear function applied in formula $\hat{y}_t = g(Vh_t)$, is the identity function, meaning that $g(x) = x$ and $g^{'}(x) = 1$. Since the cost function is a function of $\hat{y}_t$, it follows that it is really just a function of the weights that define $\hat{y}_t$. This means that the loss function lies in a multidimensional space having dimensionality equal to the number of weights in the model, which is given by the sum of the number of entries in U, V and W. A local minimum in a multivariate function is a point in which the partial derivatives of the function with respect to all the variables that define such function are equal to zero. This translates into saying that minimising the cost function just means changing the weights that define the function so that the partial derivative with respect to those weights evaluated at their actual value is as close as possible to zero.

The algorithm starts by defining $\frac{\partial C_T}{\partial V}$, which is the easiest of the three partial derivatives that will be computed. Because of the chain rule, such a derivative can be expressed as:

$$\frac{\partial C_T}{\partial V} = \frac{\partial C_T}{\partial \hat{y}_T}\frac{\partial \hat{y}}{\partial V} = (\hat{y}_T - y_T)h_T$$

Every element in the resulting matrix indicates how an infinitely small change in each of the weights collected in the weight matrix V will influence the cost function, assuming all other weights remain constant.
Next step is to do the same with respect to the matrices W and U. Let's start with the first one and let's compute $\frac{\partial C_T}{\partial W}$. By applying the chain rule, one can get:

$$\frac{\partial C_T}{\partial W} = \frac{\partial C_T}{\partial \hat{y}_T}\frac{\partial \hat{y}_T}{\partial h_T}\frac{\partial h_T}{\partial W}$$

The last partial derivative is a recursive one:

$$\frac{\partial h_T}{\partial W} = \frac{\partial h_T}{\partial z_T}\frac{\partial z_T}{\partial W} = \sigma^{'}(z_T)\left[\frac{\partial z_T}{\partial W} + \frac{\partial z_T}{\partial h_{T-1}}\frac{\partial h_{T-1}}{\partial W}\right] = \sigma^{'}(z_T)\left[h_{T-1} + W\frac{\partial h_{T-1}}{\partial W}\right]$$

it is recursive because $\frac{\partial h_{T-1}}{\partial W}$ must be computed in the same way. The partition of $\frac{\partial z_T}{\partial W}$ into two components comes from the fact that the dependency of z_T from W can be divided into an explicit and an implicit one. The explicit one is $\frac{\partial z_T}{\partial W}$ and is obtained considering h_{T-1} a constant (not influenced by W). The implicit one is the one that considers the fact that, indeed, h_{T-1} is itself depending from W and so z_T depends on W also through h_{T-1}.

The equation can be expanded because this computation can be repeated to compute $\frac{\partial h_{T-1}}{\partial W}$. This process is iterated until $\frac{\partial h_1}{\partial W}$ appears in the equation. This term needs no recursion since $\frac{\partial h_1}{\partial W} = \sigma^{'}(z_1)h_0$, where $h_0 = 0$ by definition and does not depend on W.

A final formula for the partial derivative of the cost function with respect to W can be written as:

$$\frac{\partial C_T}{\partial W} = \frac{\partial C_T}{\partial \hat{y}_T}\frac{\partial \hat{y}_T}{\partial h_T}\frac{\partial h_T}{\partial W} = (\hat{y}_T - y_T)V\frac{\partial h_T}{\partial W}$$

where $\frac{\partial h_T}{\partial W} = \sigma^{'}(z_T)[h_{T-1} + W\frac{\partial h_{T-1}}{\partial W}]$

The same approach is used to evaluate $\frac{\partial C_T}{\partial U}$, thanks to the chain rule it is possible to obtain the following equation:

$$\frac{\partial C_T}{\partial U} = \frac{\partial C_T}{\partial \hat{y}_T}\frac{\partial \hat{y}_T}{\partial h_T}\frac{\partial h_T}{\partial U}$$

Also in this case, the term $\frac{\partial h_T}{\partial U}$ can be subdivided in multiple terms. The following equation explains how:

$$\frac{\partial h_T}{\partial U} = \frac{\partial h_T}{\partial z_T}\frac{\partial z_T}{\partial U} = \frac{\partial h_T}{\partial z_T}\frac{\partial (Ux_T + wh_{T-1})}{\partial U} = \sigma^{'}(z_T)[x_T + \frac{\partial (Wh_{T-1})}{\partial U}]$$

Given that $\frac{\partial (Wh_{T-1})}{\partial U} = W\frac{\partial h_{T-1}}{\partial U}$

The equation can be rewritten as

$$\frac{\partial h_T}{\partial U} = \frac{\partial h_T}{\partial z_T}\frac{\partial (Ux_T + wh_{T-1})}{\partial U} = \sigma^{'}(z_T)[x_T + W\frac{\partial h_{T-1}}{\partial U}]$$

which is a recursive equation too and the recursion will stop when arriving at the computation of $\frac{\partial h_1}{\partial U}$ which will be simply equal to $\sigma^{'}(z_1)x1$ since h_0 does not depend on the weight matrix U. Given this equation it will be feasible to express the partial derivative of interest $\frac{\partial C_T}{\partial U}$ as follows:

$$\frac{\partial C_T}{\partial U} = \frac{\partial C_T}{\partial \hat{y}_T}\frac{\partial \hat{y}_T}{\partial h_T}\frac{\partial h_T}{\partial U} = (\hat{y}_T - y_T)V\frac{\partial h_T}{\partial U}$$

where $\frac{\partial h_T}{\partial U} = \sigma^{'}(z_T)[x_T + W\frac{\partial h_{T-1}}{\partial U}]$

Now that the gradients $\frac{\partial C_T}{\partial V}$, $\frac{\partial C_T}{\partial W}$ and $\frac{\partial C_T}{\partial U}$ have been computed, the process is iterated for M different groups of T inputs. The resulting partial derivatives will then be averaged, giving the average partial derivatives:

$$\frac{\overline{\partial C}}{\partial V} = \frac{1}{m}\sum_{m=1}^{M}\frac{\partial C_T}{\partial V}$$

$$\overline{\frac{\partial C}{\partial W}} = \frac{1}{m}\sum_{m=1}^{M}\frac{\partial C_T}{\partial W}$$

$$\overline{\frac{\partial C}{\partial U}} = \frac{1}{m}\sum_{m=1}^{M}\frac{\partial C_T}{\partial U}$$

And each weight is adjusted according to them in the following way

$$V^{'} = V - \eta\left(\overline{\frac{\partial C}{\partial V}}\right)$$

$$W^{'} = W - \eta\left(\overline{\frac{\partial C}{\partial W}}\right)$$

$$U^{'} = U - \eta\left(\overline{\frac{\partial C}{\partial U}}\right)$$

where η is a measure of how fast we want the algorithm to adjust the weights. Having η large may make the algorithm faster to converge but it may also miss the nearest minimum of the cost function or cause the algorithm to actually increase the cost of the network instead of lowering it. On the other hand, setting η low will lead to a safe and steady approach to the nearest minimum but it will require more iterations and, if too small, the training phase may remain stuck with a high training loss (Goodfellow et al., 2012). The backfitting algorithm will stop when the number of prefixed iterations is matched or when the change in loss reaches convergence, whichever of the two comes first. In modern machine learning, when dealing with standardised data, η is often chosen to be less than 1 but greater than $10e^{-6}$. (Bangio, 2012)

3.3.2 The vanishing gradient problem

By expanding the formula for the partial derivative of the cost function with respect to W, one can find an alternative way of expressing that partial derivative.

$$\begin{aligned}
\frac{\partial h_T}{\partial W} &= \sigma^{'}(z_T)\left[h_{T-1} + W\frac{\partial h_{T-1}}{\partial W}\right] = \\
&= \sigma'(z_T)h_{T-1} + W\sigma^{'}(z_T)\sigma^{'}(z_{T-1})[h_{T-2} + W\frac{\partial h_{T-2}}{\partial W}] = \\
&= \sigma'(z_T)h_{T-1} + W\sigma^{'}(z_T)\sigma^{'}(z_{T-1})h_{T-2} + W^2\sigma^{'}(z_T)\sigma^{'}(z_{T-1})\sigma^{'}(z_{T-2})[h_{T-3} + W\frac{\partial h_{T-3}}{\partial W}] = \\
&= \sum_{t=0}^{T} W^t\left(\prod_{i=0}^{t}\sigma^{'}(z_{T-i})\right)h_{T-t-1}
\end{aligned}$$

The multiplication in the middle of the formula acts on $\sigma^{'}(z_{T-i})$, but since the activation function is usually a logistic function or an hyperbolic tangent, the first derivative of σ will

be a number between 0 and 1. The result is that if the RNN is processing sequences of length T where T is big, the influence of the t time step on the gradient will become very close to zero and the neural network will only be able to adjust its weights according to the time steps that are closer to T. This makes the RNN uselessly computationally demanding and will prevent it to be able to identify relationships between observations that are far apart. This issue is called *vanishing gradient problem* (Hochreiter, 1991) and for many years it has been the main issue with RNN. In modern days, some solutions to this problem are available: the first option is to change the activation function into the Rectified linear function (ReLU), which has a first derivative that can take values that are either zero or one. A second solution to the problem is to use more advanced versions of the RNN, which use Long Short-Term Memory (LSTM) units as computational units in the hidden layer.

3.4 LSTM

LSTM networks (Hochreiter & Schmidhuber, 1997) are structured to handle long-term dependencies between observations and hence conquering the vanishing gradient problem. LSTM cells are composed by 4 elements: the cell state, the forget gate, the input gate and the output gate.

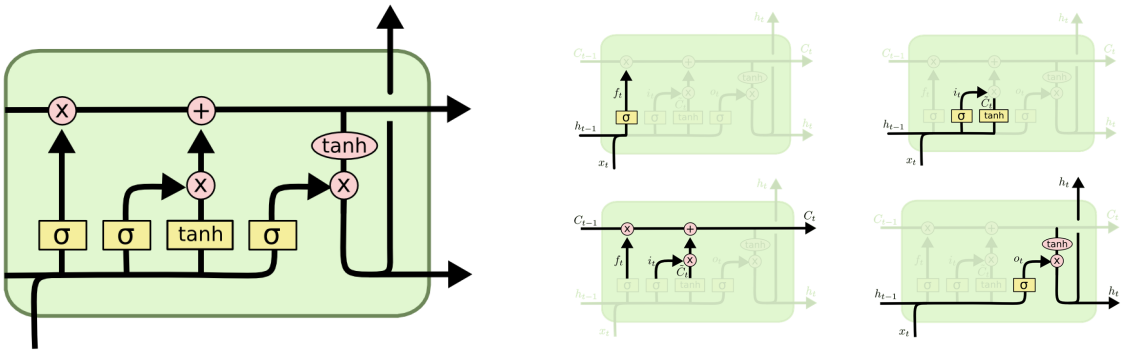

Figure 8: On the left: *Schematic representation of an entire LSTM cell.* On the right: *Highlighted components of the LSTM cell. From left to right and from the top to the bottom: forget gate, input gate, cell state update and output gate*

The cell state acts like a conveyor belt that transports relevant information through time steps, reducing the effect of the vanishing gradient. Its state at time step t is denoted with C_t and new information is stored or removed from it using the other gates. The forget gate decides which of the information in the cell state are still relevant at time step t. To do this it applies a sigmoid function to a vector obtained by computing a weighted sum of the concatenated vector made by x_t and h_{t-1}. The output will be a vector of values between 0 and 1, one for each element of C_{t-1}, in which 0 means "completely delete the information" and 1 means "completely keep the information". The process can be summarised by the following formula:

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

The input gate decides what new information will be passed to the cell state. Again, a sigmoid is applied to assign to each element of $[h_{t-1}, x_t]$ a value between 0 and 1 that will determine how much of the information will be passed. A separate operation that is done in the input layer is the transformation of $[h_{t-1}, x_t]$ through a tanh function in order to create a vector of potential candidates that may be added in the cell state. Combining the results of the two operations, a new vector will be transferred to the cell state. The operations can be summed up as follows:

$$i_t = \sigma(W_i[h_{t-1}, x_t])$$
$$\tilde{C}_t = tanh(W_C[h_{t-1}, x_t])$$

Combining the results from the forget gate to the ones from the input gate, it is possible to update C_{t-1} into C_t with the following formula:

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

The last gate is the output gate and it is here that the hidden state h_t is defined. First, again, a sigmoid function is applied to $[h_{t-1}, x_t]$ giving as a result a vector of values between 0 and 1 that will indicate which part of the cell state will become the output. The resulting vector is multiplied by hyperbolic tangent evaluated at the current cell state C_t, obtaining h_t, that will be the output of the LSTM cell at time step t. The computations in the output gate can be summed up with these formula:

$$o_t = \sigma(W_o[h_{t-1}, x_t])$$
$$h_t = o_t * tanh(C_t)$$

3.5 ARIMA and auto.arima algorithm

The ARIMA(p,d,q)(P,D,Q)$_S$ (Box et al., 2016) is a well known statistical model structured to deal with seasonalities and non stationary time series. The formula that describes the model is:

$$\Phi_P(L^S)\phi_p(L)(1-L^S)^D(1-L)^d X_t = \Theta_Q(L^S)\theta_q(L)\epsilon_t$$

where:

$$\phi(L) = 1 - \phi_1 L - ... - \phi_p L^p$$
$$\theta(L) = 1 + \theta_1 L + ... + \theta_q L^q$$
$$\Phi(L) = 1 - \Phi_1 L^S - ... - \Phi_P L^{PS}$$

$$\Theta(L) = 1 + \Theta_1 L^S + ... + \Theta_Q L^{QS}$$

and L is the Lag operator. The parameters p and q respectively indicate the number of autoregressive and moving-average terms while d indicates the number of nonseasonal differences to apply to the series. Their seasonal counterparts P, D and Q indicate the number of seasonal autoregressive terms, the number of seasonal differences to apply to the series and the number of seasonal moving-average terms. The parameter S indicates the seasonal length in the data.

3.5.1 The auto.arima function in R

The auto.arima function in R works on seasonality adjusted time series (meaning that S must be known) and follows a variation of the Hyndman-Khandakar algorithm (Hyndman & Khandakar, 2008), which starts by selecting the d parameter using several KPSS tests (Kwiatkowski et al., 1992). For the selection of the parameters p and q four models are tested: ARIMA (0,d,0), ARIMA (2,d,2), ARIMA (1,d,0), ARIMA (0,d,1). The best of the four models in terms of AICc criterion (Hurvitch & Tsai, 1989) is selected as the current model, then its parameters p and q are changed by ± 1 to give four new models to evaluate. The new models are compared with the current one using AICc again. If any of the new models has lower AICc than the current model, then it is selected as the new current model and its parameter p and q will be changed again to give four new models. This process is iterated until the current model has lower AICc than the four models generated by changing by ± 1 its parameters p and q. This whole algorithm is repeated for selecting the parameters P, D and Q. After the ARIMA model is selected, the remaining parameters are obtained through Maximum Likelihood Estimation

3.6 The baseline model

This is a very simple model that predicts future values as follows:

$$\hat{y}_{t+1} = y_t$$

It will help contextualize the performances of the ARIMA model and of the RNN.

4 Results

In this chapter of the project, the study design and the results of the experiment are presented in detail.

4.1 Study design

The Jena climate dataset has been divided into a train set containing the first 70% of the observations, a validation set containing the next 20% of them and a test set containing the remaining 10%. To have a more general idea of the behaviour of the three models, different lengths of the input sequences have been tested. This aims at simulating real life situations, in which the quantity of available data for making a prediction is not constant. As well as different lengths of the input windows, different types of output have been analysed. This project considers windows of inputs containing 24, 48 or 1000 hours of data and evaluates the behaviour of each model when applied to the forecast of both a single value and a series of consecutive of values. The single future observations predicted are the ones 1, 2, ..., 24 hours after the last observation of the input window, while the forecasted sequences are of length 1, 2, ..., 24 hours and start right after the last observation in the input window.

The baseline model has been applied to all the observations in the validation and test dataset. Applying it to the training dataset is useless because this procedure would not make the model better in terms of forecasting precision. The explanation behind this is that it has no parameters to estimate.

For the ARIMA model, a peculiar procedure had to be followed. Since the task of this project is to make predictions based on a 24h/48h/1000h window of observations, fitting an ARIMA model for each observation based on all the previous observations would have been an unfair advantage for this model predictions. To overcome the problem, an ARIMA model has been fitted for every possible 24h/48h/1000h window in the validation and test set. The auto.arima function from the forecast package in R has given the possibility to fit different kinds of ARIMA models depending on the characteristics of each window.

The RNN with 32 LSTM cells and with tanh activation function has been trained over the training set. The validation set has been used to tune the parameters of the model and to cause an early stop of the training in order to avoid overfitting. An unbiased estimate of the performances of the RNN has been computed using a test set.

4.2 Predictions of a single value at different lags

A good summary of the performances of the baseline, ARIMA and RNN model for this task is given by the computation of the loss for each prediction computed by the models. For this project, the loss function has been chosen to be the very common Mean Squared Error (or MSE), which is average value of the squared error, that is given by:

$$C_t = \frac{1}{2}(\hat{y}_t - y_t)^2$$

Where y_t is the true value of the normalised temperature in °C at time t and $\hat{y}_t$ is its forecast. An alternative could be the absolute error $C_t = \mid \hat{y}_t - y_t \mid$ but the mean squared error has been chosen because it punishes big mistakes in the predictions.

Since the RNN has been trained over the training set and has used the validation set to avoid overfitting, to avoid biases all the loss computations have been performed over the test set.

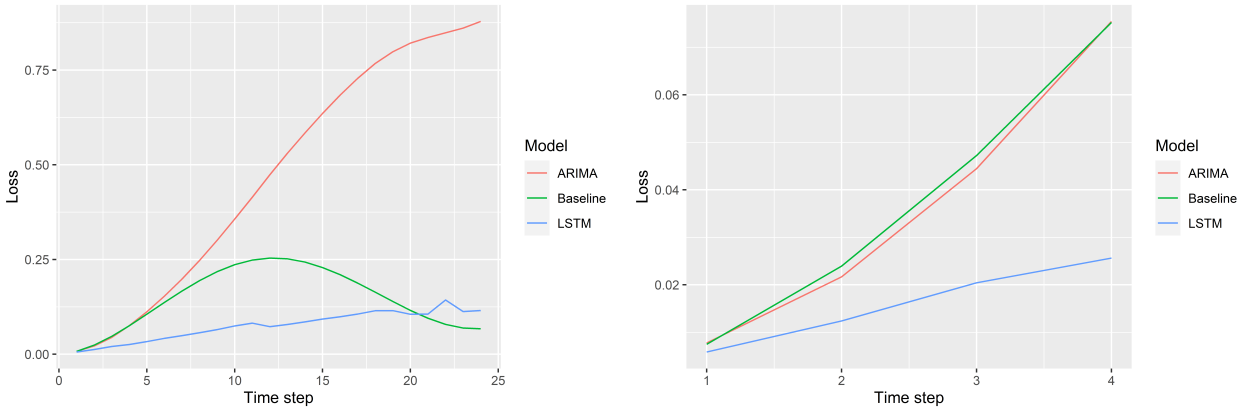

Figure 9: On the left: *graphic of the average loss of the forecasts made by the three models when predicting future observations at different time steps into the future. The models have been fitted over input windows composed of 24 hours of data.* On the right: *same graph zoomed in order to see more clearly the behaviour of the loss functions at the first four lags*

As the graph shows, the LSTM performs the best predictions for the first 20 hours, then the baseline model is able to outperform it. The ARIMA model predictions are worse than the simpler baseline model at all lags other than lags 2 and 3. Suspecting that the high value of the loss function for the ARIMA model predictions is due to the restricted size of the input window, the same experiment has been performed over windows of 48 hours length. The hope is that, given two full days of observations, the ARIMA model will be able to capture the daily seasonality of weather data and upgrade its performances.

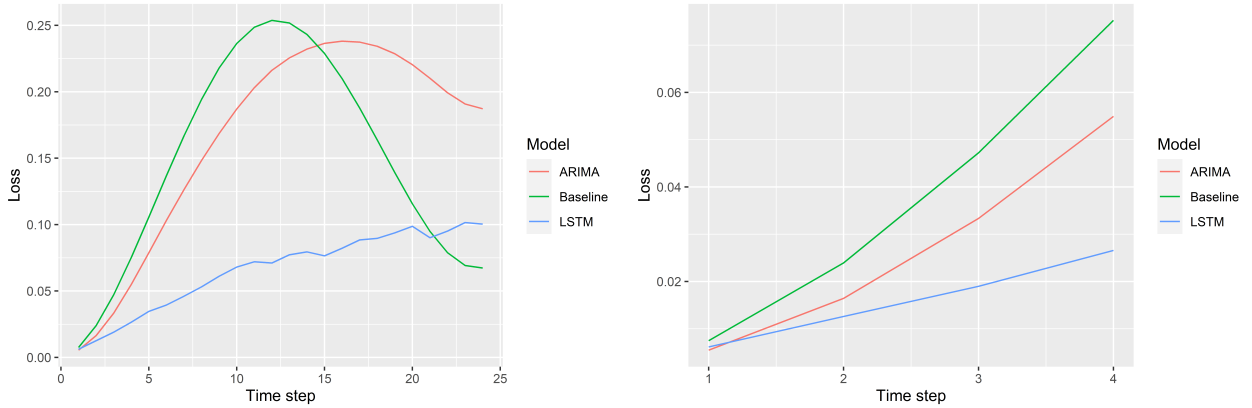

Figure 10: On the left: *graphic of the average loss of the forecasts made by the three models when predicting future observations at different time steps into the future. The models have been fitted over input windows composed of 48 hours of data.* On the right: *same graph zoomed in order to see more clearly the behaviour of the loss functions at the first four lags*

The difference in performance between the ARIMA and LSTM model has decreased just by adding 24 hours as input. While the recurrent neural network still outperforms all the others in the lags between 2 and 21, the ARIMA model predictions for the near future are getting better. ARIMA predictions are also better than the ones of the baseline model until lag 14. Seeing this rapid improve in forecasts, it is natural to wonder how much better the ARIMA model can perform if given more observations on which to fit the model and whether it can perform better than the recurrent neural network. For this reason a new simulation has been conducted with input windows containing 1000 observations.

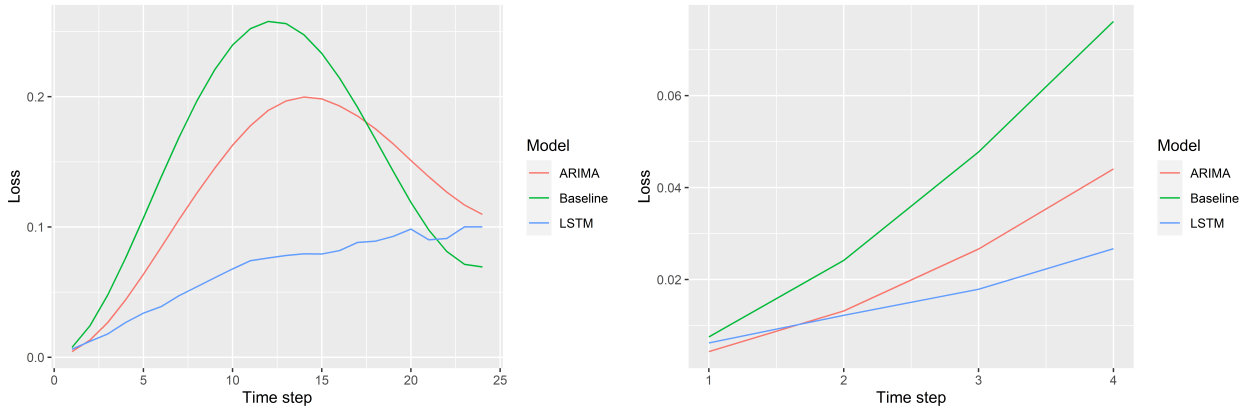

Figure 11: On the left: *graphic of the average loss of the forecasts made by the three models when predicting future observations at different time steps into the future. The models have been fitted over input windows composed of 1000 hours of data.* On the right: *same graph zoomed in order to see more clearly the behaviour of the loss functions at the first four lags*

This time the ARIMA model is by far the best model for predicting the observation at lag but, again, the RNN performs better at future lags and both models are worse than the

simpler baseline model when making predictions at 22, 23 and 24 hours from the last input observation.

A common characteristic of all the evaluated loss graphs is that the RNN is performing really good but it seems like the LSTM cell in the RNN is not able to keep in memory observations from 24h before because otherwise the graph of its loss over the 24 hours would see a decrease when approaching the 24 hours apart predictions, given the fact that observations that are 24 hours apart are similar.

4.3 Predictions of sequences of future values with different lengths

A similar procedure has been followed to establish the precision of the different models in evaluating a whole sequence of future values. The loss function has been computed as the average loss for each prediction, meaning that a resulting loss of 0.14 indicates that every forecast in the predicted sequence differed, on average, 0.14 normalised celsius degrees from the true value.

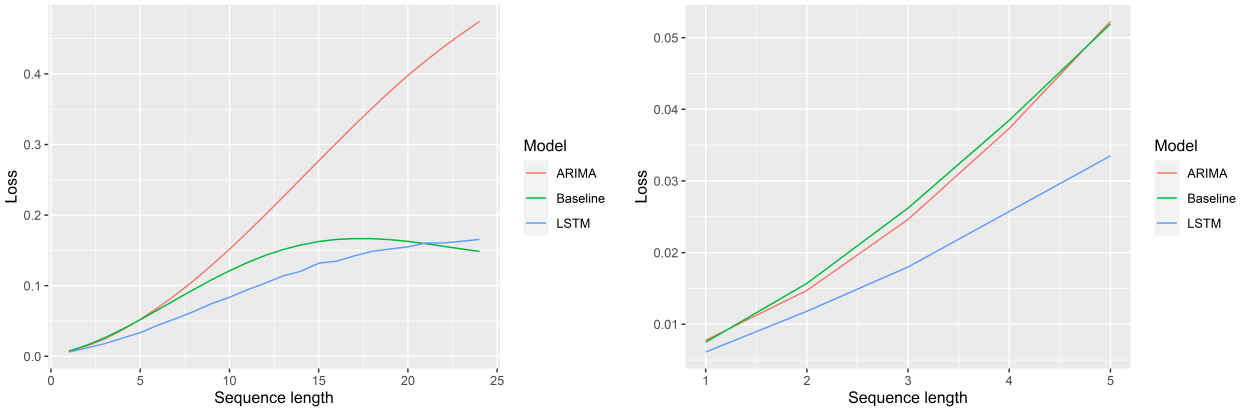

Figure 12: On the left: *graphic of the average loss of the forecasts made by the three models when predicting sequences of future observations having different length. The models have been fitted over input windows composed of 24 hours of data.* On the right: *same graph zoomed in order to see more clearly the behaviour of the loss functions at the first four lags*

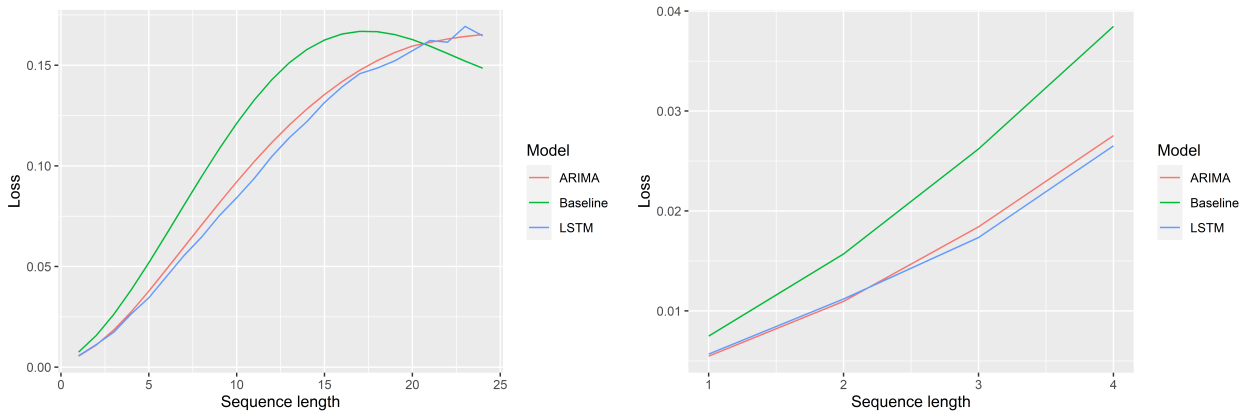

Figure 13: On the left: *graphic of the average loss of the forecasts made by the three models when predicting sequences of future observations having different length. The models have been fitted over input windows composed of 48 hours of data.* On the right: *same graph zoomed in order to see more clearly the behaviour of the loss functions at the first four lags*

Similarly to the single prediction case, the ARIMA model looks like and overfit when fitting it on just 24 observations since the baseline model is able perform better and is much simpler. The main difference from the single value estimation is that when fitting the ARIMA model over a window of 48 observations, its performances in the prediction of a sequence are already very similar to the ones of the RNN with the LSTM cell. The behaviour at lag 1 is the same as in the previous experiment since the prediction of a sequence of length 1 corresponds to the prediction of a single value 1 hour into the future.

When passing from a 48 to a 1000 observations input window, the ARIMA model is able to improve its performances. When having more data to estimate the parameters, it performs much better on average than both the baseline and the RNN with the LSTM cell in forecasting sequences of length between 1 hour and 24 hours.

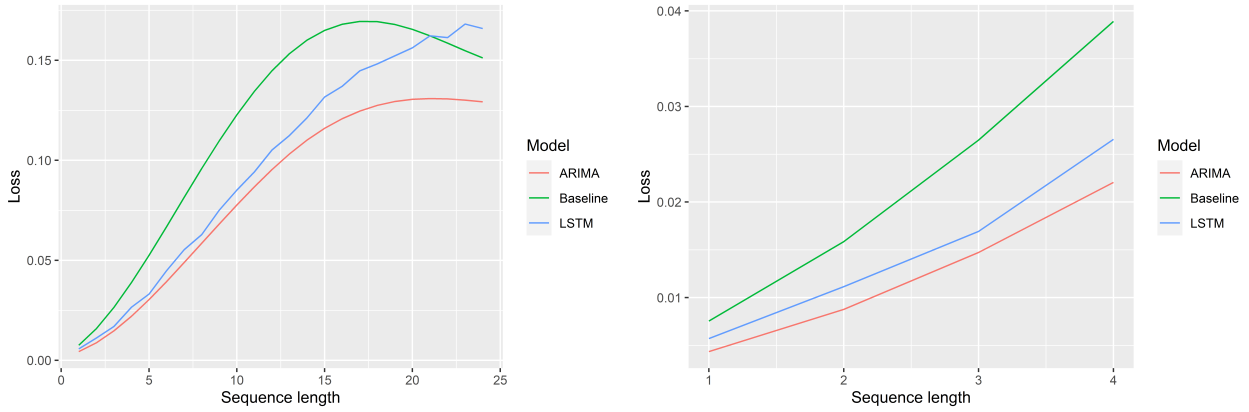

Figure 14: On the left: *graphic of the average loss of the forecasts made by the three models when predicting sequences of future observations having different length. The models have been fitted over input windows composed of 1000 hours of data.* On the right: *same graph zoomed in order to see more clearly the behaviour of the loss functions at the first four lags*

5 Conclusions

5.1 Results analysis

5.1.1 Single value prediction

From the analysis of the behaviour of the three models in this forecasting task, it is clear that a key factor in the choice of the model are the specific task to be performed and the amount of past and new data available.
If no past data are available, the recurrent neural network is not even an option since it needs several thousands of past observations to perform the training of the network. The following considerations assume that there is sufficient past information to train a recurrent neural network. In case of a total absence or an insufficient number of previous information, the same conclusions can be adjusted by not taking into account the RNN results.
The choice of the model for the single value prediction will depend on the amount of new observations available:

- 24 consequent observations: an ARIMA model is a viable option only for the prediction of the first few lags and, even for this task, its performances are similar to the ones of a much simpler baseline model. This means that using an ARIMA in this case results in an overfitting problem. The best performances until lag 20 are obtained using the RNN with a LSTM cell, while from lag 21 to lag 24 the baseline model becomes the best one.

- 48 cosequent observations: ARIMA model is preferrable for the prediction of the next observation, then the RNN is the one that performs the best predictions until lag 21. After that point, the baseline model makes the best forecasts. The main change from the previous case is that the ARIMA model performs better than the baseline model until lag 14.

- Considerable amount of new consequent observations: the ARIMA model is able to make better forecasts than the baseline model up until lag 17 and it is by far the best at predicting the next value. From lag 2 to 21 the RNN performs better than the ARIMA model and, after that point, the baseline model outperforms both of them.

5.1.2 Sequence prediction

The assumption made before about the presence of a considerably large amount of already existing data still holds.
When the task is to find the best model to predict a sequence of future observations, the answer again depends on the amount of new observations available:

- 24 consequent observations: as for the single value prediction, the ARIMA model is not able to perform as the other two models and applying it would be an overfit. The

RNN clearly performs the best predictions up until lag 20, after that the baseline model is able to make better forecasts.

- 48 consequent observations: the performances of the ARIMA model and of the RNN are very similar. While the ARIMA model seems to be the best at predicting the observations at lag 1 and 2, the RNN performs slightly better from lag 3 until lag 21. After that, the baseline model outperforms both of them.

- Considerable amount of new observations: in this case the ARIMA model is able to outperform both the baseline model and the RNN from lag 1 up until lag 24 and hence it is the best model for predicting sequences of values.

5.2 Considerations and weaknesses of the experiment

5.2.1 The baseline model

This easy model is suitable for every situation because it does not need previous observations to train. It also needs just one new observation to be able to make a prediction, as opposed to the ARIMA model. An alternative that can be used is a median model, for which the forecast at every time step is simply the median of the time series.

5.2.2 The ARIMA model

The main consideration regards the uncertainty level of the predictions: both the baseline model and the RNN predictions are not associated with any uncertainty level, while for the ARIMA model confidence intervals can be generated using the same auto.arima function used to fit the models. Moreover, this model is structured to deal with time series of a single variable, multivariate versions of the ARIMA such as the Vector autoregression model (Sims, 1980) may perform better.

5.2.3 The RNN

An important aspect to keep in mind when analysing the results is that recurrent neural networks are trained with a specific purpose. This means that a RNN trained to perform the best predictions at lag 1 will perform far from the optimal if used to perform predictions at time step 2. For this reason different RNNs have been trained in order to obtain the best predictions at a given time step. This is extremely computationally demanding and it has been possible because this project has been developped over the course of several months. The training of a RNN may take from 1 hour up to 24 hours, depending on the length of the input sequence that is fed into the network. With the input window containing 1000 observations the training required an entire day of computations and this time consuming training is not always possible. Moreover, the RNN is the only one of the three models that uses the whole dataset, all the other ones only use the variable describing the temperature.

Another important aspect is that nowadays with a powerful computer it is possible to train RNN with deeper and more complex structures, which will probably perform better than the one trained for this project. The RNN used in this experiment seemed to perform worse when predicting values that are far from the last input observation, more recent RNN structures allow the network to improve its memory through time and so they should be able to overcome this problem.

5.3 Validity of the results outside of this project and suggestions for future work

These results are valid when applied to time series of similar shape. If the sequence under analysis presents sudden spikes or lowerings, then the performances of the models may vary. Future work on this subject may include the analysis of different time series and the fitting of more complex models such as the VARIMAX and Vector autoregression model or a RNN with a more complex structure and different activation functions.

6 Peer review reflections

Both the peer reviewers gave me important suggestions to improve my project. A common advice was to improve the presentation of the graphs in the report by adding labels under the figure. I did this trying to be as exhaustive as possible when writing them, I also added legends in the Results section to make the graphs easier to understand. Peer reviewer 1 suggested me to make the introduction of the models more aesthetically pleasing, it has been a helpful tip and I applied it by using a bucket list and by writing in bold the names of the three models, while trying to keep the explanations as clear as possible. The reviewer also suggested numbering the headers and making the titles larger, which I found to be a good idea: it makes the report tidier and much easier to follow. Peer reviewer 2 pointed out that the correlation plot created with ggpairs in page 6 is hard to read and that he would have reduced the variables in the graph or even substitute such graph with a correlation matrix. The decision that I took was to reduce the number of variables, keeping only the ones that show clear correlation. I think that using a correlation matrix might be a good approach with different datasets. In the Jena climate dataset all variables are weather variables and sometimes they present correlations that are different from a linear one. Having a variable against variable plot to observe may help to spot these non-linear correlations. Peer reviewer 2 also pointed out that it was a shame that there were not more sections about the RNN. What I submitted for the peer review was a file containing only the Introduction and the Data manipulation and exploratory analysis chapter. I added a full description of the RNN in the Methodology chapter, which has been written after the peer review process. This peer review process helped me improve the project, in particular in terms of presentation of my work. The first draft of the project that I submitted for the peer review was poorly presented, looking at other people's work made me realise the importance of an aesthetically pleasant report. Moreover, reading about unknown subjects highlighted the value of a good introduction, which is a section that I overlooked in the beginning.

References

Bangio, Y. (2012). *Practical recommendations for gradient-based training of deep architectures.*

Box, G. E. P., Jenkins, G. M., Reinsen, G. C., & Ljung, G. M. (2016). *Time series analysis: Forecasting and control* (Fifth edition).

Goodfellow, I., Bengio, Y., & Courville, A. (2012). *Deep Learning.*

Hochreiter, S. (1991). *Untersuchungen zu dynamischen neuronalen Netzen.*

Hochreiter, S., & Schmidhuber, J. (1997). *Long-Short Term Memory.*

Hurvitch, C. M., & Tsai, C.-L. (1989). *Regression and time series model selection in small samples.*

Hyndman, R. J., & Khandakar, Y. (2008). *Automatic time series forecasting: The forecast package for R. Journal of Statistical Software.*

Kwiatkowski, D., Phillips, P. C. B., Schmidt, P., & Shin, Y. (1992). *Testing the null hypothesis of stationarity against the alternative of a unit root.*

McCulloch, W., & Pitts, W. (1943). *A Logical Calculus of Ideas Immanent in Nervous Activity.*

Mozer, M. C. (1995). *A Focused Backpropagation Algorithm for Temporal Pattern Recognition.*

Nair, V., & Hinton, G. (2010). *Rectified linear units improve restricted boltzmann machines.*

Pearson, K. (1896). *Mathematical Contributions to the Theory of Evolution. III. Regression, Heredity and Panmixia.*

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). *Learning Internal Representations by Error Propagation.*

Sims, C. (1980). *Macroeconomics and Reality. Econometrica.*

Tukey, J. W. (1977). *Exploratory data analysis.*