

École polytechnique de Louvain

Prediction of the number of new coronavirus-related hospitalizations based on open source data

Authors: **Augustin DELECLUSE**, **Margaux GÉRARD**

Supervisor: **Pierre SCHAUS**

Readers: **Guillaume Derval**, **Nicolas Golenvaux**, **Pierre Schaus**

Academic year 2020–2021

Master [120] in Computer Science and Engineering

PREDICTION OF THE NUMBER OF NEW
CORONAVIRUS-RELATED HOSPITALIZATIONS BASED ON
OPEN SOURCE DATA

Augustin DELECLUSE & Margaux GÉRARD

*June 13, 2021
École Polytechnique de Louvain
Rue Archimède 1 - Bte L6.11.01
B-1348 Louvain-la-Neuve*

Acknowledgements

First of all, we would like to express our gratitude to Professor Pierre Schaus, who has been our thesis supervisor during this last year. Thanks to his advice, feedback and his implication in open-source data related to the coronavirus in Belgium, we have been able to complete this document. We hope that our work lives up to his expectations.

Additionally, we would like to express our thanks to Guillaume Derval, who has written a paper about the estimation of the number of hospitalizations related to the coronavirus using *Google Trends* [1]. Based his work, we have been able to quickly build working prototypes when using *Google Trends* data.

Moreover, we would like to thank all the professors we have crossed paths with during our 5 years of study at *École Polytechnique de Louvain*. Their teaching has allowed us to grow intellectually and develop as young engineers. The quality of the academic training we have received allows us to feel confident regarding our evolution and our future career path.

A special mention to Martin Delecluse, François De Keersmaecker and Benoît Michel for their careful proofreading and advice. We thank you warmly for your precious help, your kindness and your availability.

Last but not least, we would like to thank our respective families and friends for their unfailing encouragement and support without which this work would not have been the same.

Abstract

The health crisis due to the COVID-19 that we have now been experiencing for over a year has affected over 175 000 000 people and killed over 3 800 000 people around the world so far [2]. Could we have lowered these numbers if we had been able to predict the outbreak of the pandemic accurately?

Being able to predict the number of hospitalizations due to COVID-19 in our country accurately would have been very useful in recent months. Indeed, it could have allowed our leaders to take difficult decisions such as announcing a new lockdown more quickly and thus avoid saturation of hospitals before it being too late. This is why we decided to develop an artificial intelligence prediction model based on open data, namely the number of hospitalizations in Belgium (and a few other regions of interest) and *Google Trends* data. The aim of our work is also to determine to what extent the data collected via *Google Trends* can help us refine a basic short-term prediction.

Abbreviations

LSTM	Long Short-Term Memory
MSE	Mean Squared Error
MAE	Mean Absolute Error
RMSE	Root Mean Squared Error
ReLU	Rectified Linear Unit
ELU	Exponential Linear Unit
RMSprop	Root Mean Square Propagation
ANN	Artificial Neural Network
RNN	Recurrent Neural Network
ICU	Intensive Care Unit
API	Application Programming Interface
IP	Internet Protocol
NaN	Not a Number
GPU	Graphical Processing Unit
LR	Linear Regression
E-D	Encoder-Decoder

Contents

Acknowledgments	ii
Abstract	iii
Abbreviations	iv
1 Introduction	1
2 Data collection	3
2.1 Collection of the number of hospitalizations	3
2.1.1 Hospitalizations in Belgium	3
2.1.2 Hospitalizations in France	4
2.1.3 Hospitalizations in the world	4
2.2 Collection of <i>Google Trends</i>	4
2.2.1 Search terms and topics	5
2.2.2 Topics related to the coronavirus pandemic	5
2.2.3 Sampling of the <i>Google Trends</i> data	7
2.2.4 Combining several requests	8
2.2.5 Problems encountered	10
2.2.6 Different collection methods	16
3 Data preparation	19
3.1 Data preprocessing	19
3.1.1 Rolling average	20
3.1.2 Removing NaN values	20
3.1.3 Feature scaling	20
3.2 Data transformation	22
3.2.1 Logarithmic transformation	22
3.2.2 Percentage change transformation	23
3.2.3 Shifting	23
3.3 Data augmentation	26
3.3.1 For France/Belgium	26
3.3.2 For Europe	27
4 Background	29
4.1 Artificial Neural Network	29
4.1.1 Components of an Artificial Neural Network	29
4.1.2 Learning	34
4.2 Architecture of a Recurrent Neural Network	38

4.3	Architecture of a Long Short-Term Memory	39
4.3.1	Dropout and recurrent dropout	41
4.4	Architecture of a Bidirectional Long Short-Term Memory	42
4.5	Architecture of an Encoder-Decoder	43
5	Prediction models	45
5.1	Reference models	45
5.1.1	Linear regression model	46
5.1.2	Repeat Last model	46
5.1.3	Dense model	47
5.2	First approach	48
5.2.1	Tuning of hyper-parameters	50
5.3	Second approach	51
6	Performance analysis	53
6.1	Evaluation metric	53
6.2	Feature importance	56
6.2.1	Based on sensitivity analysis	56
6.2.2	Based on mean decrease in impurity	59
6.2.3	Based on feature permutation	60
6.3	Prediction of the new hospitalizations	61
6.3.1	Reference models	62
6.3.2	Encoder-decoder model	68
6.3.3	Assembled model	73
6.3.4	Comparison of each model	76
6.4	Real-time prediction	77
6.4.1	Installation	78
6.4.2	Utilization	79
7	Discussion of the results	82
7.1	Data collection	82
7.2	Feature importance	84
7.3	Prediction	85
8	Conclusion	88
	Bibliography	95
	Appendices	96

February the 4th, 2020. The first confirmed case of coronavirus has been detected in Belgium. Gradually since then, the number of confirmed cases have gone up, increasing rapidly in March and April, leading Belgium to reach the third highest number of deaths due to COVID-19 per inhabitants worldwide in March. Against the spread of the virus, the Belgian government ordered several measures, such as the closing of multiple activities: schools, bars, gathering and others. Those measures had a lot of consequences on the population and the daily life. Those impacts can be economic [3], with the tourism industry registering a loss of more than a billion euros [4], or even psychological [5], suicidal thoughts and attempts even rising up in our country. Those decisions taken to limit the spread of the virus in Belgium are currently guided by mathematical prediction models that are based on a large number of assumptions. The scenarios considered are therefore very dependent on these assumptions. There are also many parameters that have been neglected since it is not possible to take all factors into account in those types of model. Because of this, the relevance of such models can be questioned as some of the parameters used in them are set arbitrarily. For example, the contagiousness of the UK variant could be 30%, 50% or 70%. However, depending on the value of the contagiousness that is used in the models, the final scenario will not be the same [6].

Generally speaking, measures taken by the government are decided on the basis of the current spread of the virus. Being able to predict it correctly could help to prepare better counter actions, potentially save lives and have huge economic impacts. To quantify the spread of the coronavirus, several statistics are important regarding infected people, but some have flaws or their relevance has changed over time. For instance, the number of cases detected is dependent on the testing that is done, and fewer tests were available at the beginning of the crisis. The number of deaths is also important but some critics have been made regarding those numbers, notably because the number of deaths reported in retirement houses may have been overestimated, as they include deaths that are considered linked to the COVID-19 even if not proven by a test [7], a choice that was not made in every country. But the number of hospitalizations kept relatively the same importance over time. Indeed, it is not influenced by the number of tests available. Furthermore, it indicates how many patients can still be accepted in a hospital, as the number of remaining beds available is dependent on it.

In this master's thesis, we will try to predict as accurately as possible the future number of hospitalizations happening in Belgium. Those predictions will be based on the past number of hospitalizations as well as *Google* searches related to the coronavirus. Including internet searches for refining a prediction is not a new idea: *Google* developed such a

system several years ago, named *Google Flu Trends* [8]. It aimed at predicting flu activity in several countries. The system proved to be efficient at the beginning, but failed to predict accurately the number of doctors' visits thereafter, even predicting twice as many visits as actually recorded. For those reasons, the system was stopped in 2015.

Using those data, we will try to answer the following questions:

- Is it possible to accurately predict the number of new hospitalizations caused by the coronavirus?
- Can *Google* searches be used to enhance those predictions?
- What are the topics searched on *Google* that have the highest impact on predictions?
- Can this prediction system be generalized to other countries than Belgium?

To answer those questions, we will first explain in depth the data set used, as well as collection techniques relative to it. Then, we will explain some processing done on the data, in order to feed it into a prediction system. After that, we will explain the theoretical background of the techniques used, along with several prediction models based on it. The predictions done by those models will then be evaluated, and the impact of the *Google* searches given as input will be quantified. Finally, we will discuss the results, in order to evaluate if one of those prediction model can be used with a high confidence, as well as some practical considerations.

Before we can make any predictions about the future, we need to know past data in order to feed a prediction model. We aim to predict the hospitalizations in Belgium, but collecting the data for this country only might not provide enough information to work on. We have thus also considered other regions, that will be used to feed more data into our prediction system and to assess its performance.

We have collected two distinct types of data. On the one hand, we collected data related to the number of hospitalizations due to COVID-19 in Belgium as well as in other regions. On the other hand, we also collected open-source data through the *Google Trends* tool [9] in order to find out how often a term has been typed into the *Google* search engine. We have chosen to use *Google Trends* as this data is freely available in most countries of the world. This chapter will be devoted to the description of the data collection we have carried out.

When we talk about the number of hospitalizations in general, it refers to the number of total hospitalizations due to COVID-19 as well as the number of new hospitalizations, i.e. the variation of the total number of people who are at the hospital. The total number of hospitalizations represents the number of people who are at the hospital at any given time. The number of new hospitalizations is the difference between the number of people admitted to hospital and the number of people discharged from hospital at a particular time. In this report we will use both types of hospitalizations data but not in the same way. Indeed, the number of new hospitalizations is the target that we wish to predict while the number of total hospitalizations is a feature that will be used to help the prediction model to provide satisfactory results.

2.1 Collection of the number of hospitalizations

Because the first hospitalizations due to COVID-19 were reported in the middle of March 2020 in Belgium, we have assumed that the number of hospitalizations was equal to zero before this date. The same process is applied to all regions: the hospitalizations before their first reported date was considered as equal to zero.

2.1.1 Hospitalizations in Belgium

The daily number of hospitalizations is retrieved from the *Epistat* website [10], hosted by *Sciensano*, the Belgian institute for health [11]. This dataset is updated daily and contains information about the number of confirmed cases, hospitalizations, deaths and tests done relative to the coronavirus pandemic. Some concerns were raised about this dataset

[12], complaining about a lack of transparency: for instance the data relative to the number of tests done is not available by age while *Sciensano* mentions it in some of its reports.

For this dataset, we only used the daily total number of hospitalizations and new hospitalizations. These values are relatively important during the coronavirus crisis. Indeed, hospitals are limited in terms of the number of beds and respiratory assistance devices available. It is therefore imperative to limit the number of people who are hospitalized at the same time to avoid saturating the capacity of hospitals. Secondly, we would have included the number of case detected in the predictive model. However, this number is unstable as it depends on the amount of tests done. In addition, the amount of tests available at the beginning of the crisis was limited and therefore the number of cases detected may not be representative of the reality. Indeed, at that time, people were tested in Belgium only if they showed symptoms of the coronavirus or if they had been in close contact with a person who tested positive to COVID-19.

2.1.2 Hospitalizations in France

The daily number of hospitalizations is retrieved from the site of *data.gouv.fr*, the open platform for French public data [13]. Compared to *Sciensano*, the French dataset contains more entries, such as the number of tests done by gender or the number of people who entered the intensive care unit (ICU).

2.1.3 Hospitalizations in the world

Most of our analysis is based on the data available in Belgium and France but the methods described in this document can also be applied at the scale of countries rather than French regions and Belgium. We have therefore also collected data relative to European countries. Some results relative to it can be found in the appendices, as our objective remain the prediction of hospitalizations in Belgium and not in every country.

The daily number of hospitalizations is retrieved from the site *Our World in Data* [14]. It contains data relative to the number of hospital admissions as well as ICU occupancy, number of positive cases, vaccines and various indicators about several countries in the world. This dataset was used instead of the one from *European Centre for Disease Prevention and Control* [15] because it includes data relative to more countries, even in Europe. For instance, data from Great Britain were not present in the dataset from *European Centre for Disease Prevention and Control* but are well present in the one that we use.

2.2 Collection of *Google Trends*

Google Trends [9] is a tool from Google that allows us to find out how often a term has been typed into the Google search engine, with the possibility of viewing this data by region and by language. This data is represented as a relative integer value between 0 and 100, 100 being the period where the most queries related to the given term occurred and

0 noting that no data could be retrieved for this period. *Google Trends* does not provide an official API but some unofficial ones exist, such as *pytrends*[16], which allows to make queries to the *Google Trends* website using Python.

To convince you of the simplicity of using *pytrends*, let us show you an example. Imagine that the goal is to get the interest over time of the topic "*Anosmia*"¹ in Belgium for the previous three months. The following code can be run:

```
1 # Import
2 from pytrends.request import TrendReq
3 # Connection to Google Trends server
4 pytrends = TrendReq(hl='en-US', tz=360)
5 kw_list = ["Anosmia"]
6 pytrends.build_payload(kw_list, cat=0, timeframe='today 3-m', geo='BE')
7 # Get interest over time for the keyword "Anosmia"
8 print(pytrends.interest_over_time())
```

Listing 2.1: Example of the use of *pytrends*

2.2.1 Search terms and topics

The study of the interest over time of queries can be done thanks to a search term or a topic. Topics will include all search terms that are related to it regardless of the language that is used. Search terms are specific and results will only show the relative frequency of occurrence of a specific term in the search engine. In other words, topics are a collection of search terms. To give a simple example, the search term "Fever" will only consider the interest over time of queries that contain the word "Fever". In contrast, the topic "Fever" will also contains search terms like "Fièvre" (translation in French), "Body temperature", "sweat", etc.

We have chosen to work with topics instead of search terms for three main reasons. The first one is that topics are not subject to a specific language, which comes handy in Belgium where there are three official languages. Moreover, a topic belongs to a given category (*Health, Disease, ...*) and this can be useful in order to ensure that the interest over time is that of the topic in the context being researched. Indeed, two words can be written in a semantically similar way but used in different contexts. In this case, the two words will belong to different categories. For example, the word "testing" may refer to coronavirus testing (the one being targeted), software testing or product testing. Finally, the evolution of the interest over time of a topic will be, most of the time, much smoother than the one of a search term.

2.2.2 Topics related to the coronavirus pandemic

Now that we know that we will be querying *Google Trends* with topics, we need to determine which topics are most relevant, i.e. most related to the number of hospitalizations due to the coronavirus. To do this, we decided to choose topics related to symptoms

¹Decreased or complete loss of smell

of the coronavirus [17]. This way, we can hope that someone suffering, for instance, from agueusia² has searched for this term or a term relative to that topic in the *Google* search engine. Therefore, a first list of symptoms linked to the coronavirus was produced. This list was built upon the check-lists that doctors were using in Belgium to detect the coronavirus cases at the beginning of the crisis. The symptoms were written in French and include:

- | | | |
|---------------------------|-----------------|----------------------|
| • Fièvre | • Mal de gorge | • Essoufflement |
| • Toux | • Courbatures | • Rhinite |
| • Diarrhée | • Nez qui coule | • Agueusie |
| • Difficulté respiratoire | • Mal au ventre | • Céphalées |
| • Perte gout | • Mal au dos | • Anosmie |
| • Perte odorat | • Maux de tête | • Vomissements |
| • Fatigue | • Perte appétit | • Douleur abdominale |

Based on this list, we built the different topics that will be used. Those topics were also presented on the *Google Trends* platform as the most searched symptoms linked to the coronavirus:

- | | |
|-----------------------|-----------------|
| • Fever | • Loss of taste |
| • Sore throat | |
| • Shortness of breath | • Loss of smell |

In addition to retrieving interest over time, *Google Trends* also allows to find related queries and topics with respect to a given keyword. We have recovered both queries and topics related to the initial symptoms and topics, retrieved their evolution over time and compared it to the number of hospitalizations each day by using the *Spearman* correlation value [18]. The keywords with a correlation ρ such that $|\rho| \geq 0.65$ were kept and constituted a new set of keywords used to find additional related queries and topics related to the coronavirus.

One should note that keywords retrieved from this process might not necessarily be symptoms of the coronavirus, they could be any word. For example, at the beginning of the crisis, we were surprised to notice that the word "coronavirus" was highly correlated with the word "puzzle", which was puzzling to us as well !

The final set of topics that we have chosen to use includes:

²Decrease or total loss of taste

- Allergy
- Coronavirus disease 2019
- COVID 19 testing
- COVID 19 vaccine
- Cure
- Fever
- Sore throat
- PCR
- Symptom
- Cough
- Vaccination
- Virus

Some initial topics were dropped as a low amount of data related to it was found or as their correlation was not high enough. Note that when we started work on this thesis, we did not include terms related to vaccination and testing. These terms emerged as important later. We will discuss them in more detail in section 6.2 of this report.

2.2.3 Sampling of the *Google Trends* data

There are two possible samples of data that can be accessed through *Google Trends*: real-time and non-real-time data. The first one is a deterministic sample that covers the last seven days on an hour-by-hour basis. With this type of sampling, most of the intervals dating more than 7 days can also be recovered, but there may be missing data. The second data sampling is a stochastic sampling that can go back from 2004 to more or less 36 hours ago. The sampling done for this second type depends on the time interval asked: it can be sampled on a daily, weekly or monthly basis. In fact, each query made over a period of time of less than nine months will be sampled on a day-by-day basis.

During our work, we eliminated week-by-week sampling because it contained little data. Indeed, the coronavirus pandemic started in March 2020 in Belgium, which means that we would have barely 65 data points for each topic and each region between February 1st 2020 and June 13th 2021. The next step was to decide whether to work with a hour-by-hour or a day-by-day data sampling. Since the hospitalization data are sampled on a daily basis, we would like to collect the *Google Trends* data on a daily basis as well. Nevertheless, it is fairly simple to resample hourly data to daily data. Therefore, both data sampling can be considered for this purpose. Table 2.1 shows the main differences between each of the two sampling. Note that the items in bold are the ones we are looking for.

	Hour-by-hour	Day-by-day
Newest data	Last hour	36 hours earlier
Randomness	Deterministic	Stochastic
Maximal period for request	7 days and 23 hours	9 months
Missing batches of data	Sometimes	Never

Table 2.1: Comparison between sampling on a hour-by-hour or a day-by-day basis

This means that hourly data could be interesting because it allows us to create a real-

time prediction model with the freshest data from the previous hour. Moreover, queries are deterministic, which means that a query over a period of time will always produce the same results, no matter when or how it is performed. However, no matter which sampling is chosen, the maximum time interval that can be queried is limited. Indeed, for hourly data the maximum time interval that can be queried is a little bit more than seven days, while it is nine months for daily data. Luckily, there are ways to combine several requests in order to cover as many time intervals as needed. The method that we have chosen to use will be described in the next section.

2.2.4 Combining several requests

In order to be able to collect data to cover a period of more than one year without getting weekly sampled data, we need to combine several requests (daily or hourly). But because the data given by *Google Trends* is relative, this means that we need to be careful when combining the requests. As an example, two queries done on adjacent time laps are shown on the next Figure:

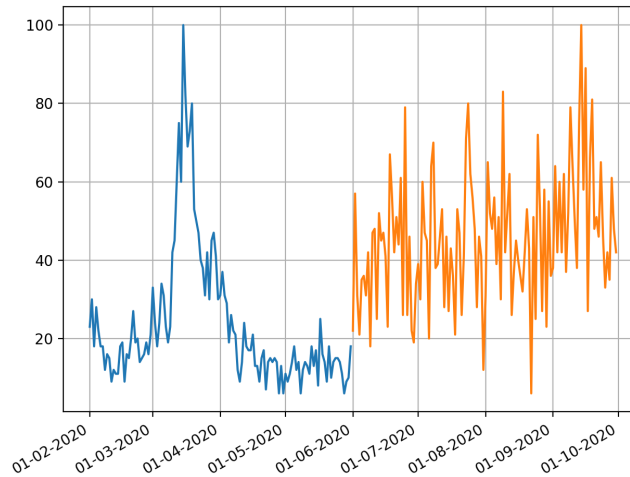


Figure 2.1: Two daily queries on adjacent time interval

As we can see, values of two queries cannot simply be concatenated to form an aggregated query. Indeed, doing so would result in having the value 100 appearing twice for the same signal but without guarantee that those occurrences represent the same number of requests done. We have therefore chosen to use an overlap between queries covering adjacent timelines. From the values of each query on this overlap interval, we can compute which request contain the highest values and scale the signal of the other query accordingly. Because this method is not dependent on the sampling of the data, it can be performed on both daily queries (Figure 2.2) and hourly queries (Figure 2.3). Note that we used a mean of several daily queries to compensate their stochastic nature and to reduce the noise of the results returned by *Google Trends*. More explanations about the mean of daily queries are given in section 2.2.5.

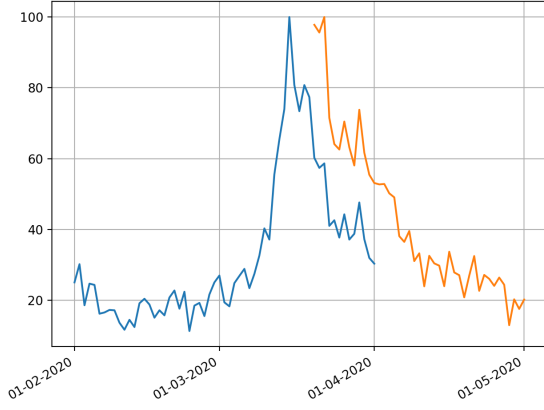


Figure 2.2: Two daily queries with an overlap. Each query was retrieved using an average of 20 daily queries

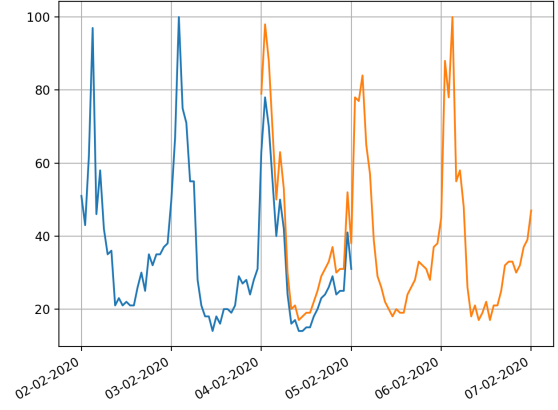


Figure 2.3: Two hourly queries with an overlap

When computing the scaling on this overlap, the values equal to zeros in those queries are ignored, as this corresponds to a lack of data according to *Google Trends*. In both cases (Figures 2.3 and 2.2), the two neighboring queries represented were chosen such that their total time span can be covered using only one daily request in the *Google Trends* tool. Therefore, after merging and scaling the two queries, we can compare the values given directly by *Google Trends* on this interval with respect to the merged query that we have computed. This can be seen on Figures 2.4 and 2.5.

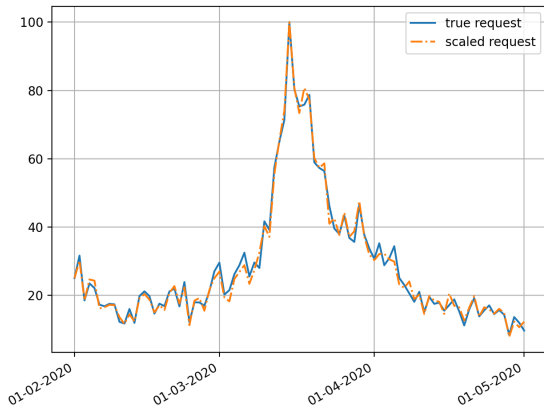


Figure 2.4: Signal obtained by merging two daily requests, along with the signal given by *Google Trends*. Each query was retrieved using an average of 20 daily queries

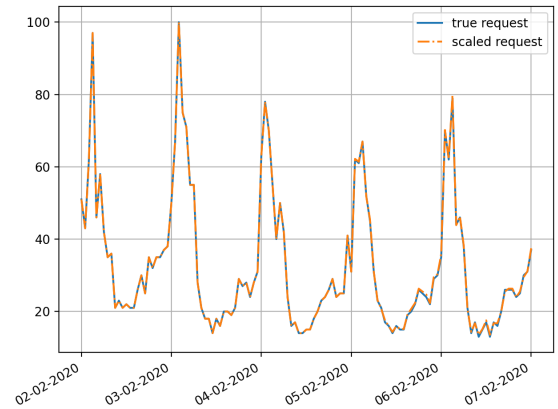


Figure 2.5: Signal obtained by merging two hourly requests, along with the signal given by *Google Trends*

In both cases, the signal computed with our method for combining several requests is extremely close to the values given by *Google Trends*. Figure 2.4 shows more differences between both signals than Figure 2.5. This is due to the small variations resulting from the stochastic nature of the daily queries, even though we averaged the signal using several queries to reduce the noise of the results returned by *Google Trends*.

2.2.5 Problems encountered

Stochastic requests

Intuitively, we thought about working with data sampled on a day-by-day basis because it was simpler. Indeed, the prediction model that we will describe later tries to predict data several days ahead and it is therefore preferable that the data is already sampled at the right scale, i.e. in days. As a reminder, the data collected on a daily basis is stochastic. We therefore tried to quantify the variations to make the signal more reliable. To do this, we investigated the possibility of combining several queries done on the same interval, in order to see if they converge to the same value when averaged. Figure 2.6 shows the signal obtained when using a 100-queries average for each daily sample.

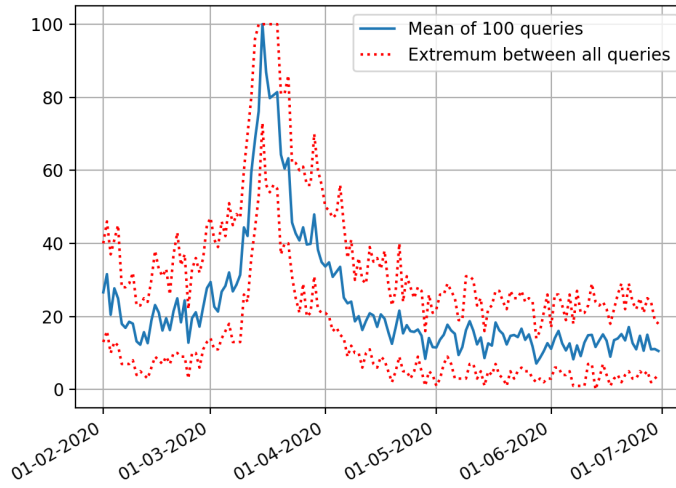


Figure 2.6: Mean of one hundred daily queries, along with the maximum and minimum on each day between all queries. The mean is rescaled between 0 and 100, to corresponds to the usual format of the data

Figure 2.7 shows the convergence of the error depending on the number of queries aggregated. The error is defined based on the difference between the values computed when using an average of x and queries and when using an average of $x - 1$ queries. We observe that we do not absolutely need to take the mean of a hundred requests, as little is gained by taking more than the mean of 20 queries. By using 20 queries, we get a quasi deterministic dataset, i.e. a dataset that will make an absolute error small enough to be considered as deterministic. We also observe from Figure 2.6 that results from two similar requests can vary a lot, and that those

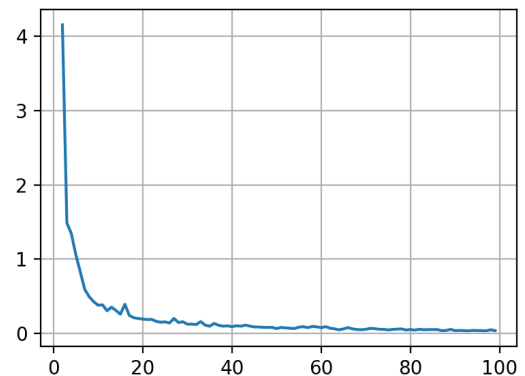


Figure 2.7: Absolute error in function of the number of queries used for the mean

variations needs to be considered.

Unfortunately, we cannot simply send a query with the same arguments over and over in order to get the average of several daily interest over time values. Firstly, *Google* is more prone to return errors when the same query is requested repeatedly. Secondly, *Google* seems to cache its last asked queries, in order to provide a result rapidly. As we cannot change the topic nor the localization of the query, we have decided to use tiny lags within our time span requested while ensuring that the targeted period of time was always fully covered by the query. This leads to queries with different parameters, forcing *Google* not to use its cached data. For example, for collecting the daily data covering the time period between 24 February and 17 March, we could do these requests:

- From 24 February to 17 March
- From 23 February to 17 March
- From 24 February to 18 March
- From 23 February to 18 March
- From 22 February to 17 March
- ...

With this method, we ensure that the data are not stocked in the cache during the collection. Moreover, the period of time that we want to retrieve (from 24 February to 17 March) is always present in the daily query that is done. A representation of the iterations using a timeline is also shown on Figure 2.8. In this representation, we also observe that the time interval between a and b is always covered by the queries.

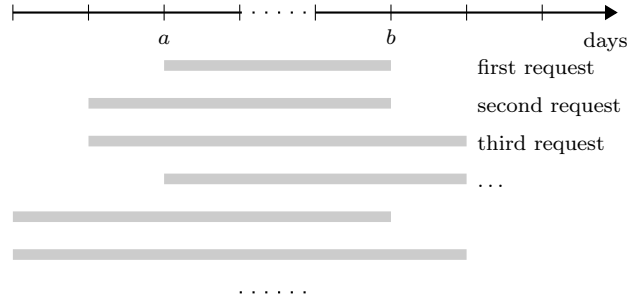


Figure 2.8: Iterating over several time interval to provide daily requests over a , b . The iteration pattern minimizes the number of additional days needed, as the length between a and b is limited by the length of the largest query used, which cannot exceed 9 months.

This method to get daily data which is quasi-deterministic works, but is hard to use with a lot of regions and topics. Indeed, for the data collection, we consider 22 French regions in addition to Belgium, which makes a total of 23 regions on which we have to collect data. This means that we need to make 6900 requests ($23 \text{ regions} \times 15 \text{ topics} \times 20 \text{ requests}$ to be deterministic) each time we need to collect new data, which is really too much. Moreover, it is important to remember that daily data is harder to use with a real-time prediction model since it only allows us to get data from 36 hours ago or more. That is why we have explored the possibility of using hourly data, which also has some drawbacks.

Missing batches of data

One of the biggest issue that we faced with data sampled on a hour-by-hour basis was that some batches of data were missing. This means that *Google Trends* was not able to provide us with data for the query that was made. On Figure 2.9, which represents hourly batches of data, we observe that some batches have a constant value of interest of zero. For example, there are three consecutive missing batches in March. By checking this information on the *Google Trends* platform, we note that no data is available between February 24 and March 17. Because we need an overlap between batches of data to scale it correctly, we need a way of estimating the values of search *trends* in these missing data areas.

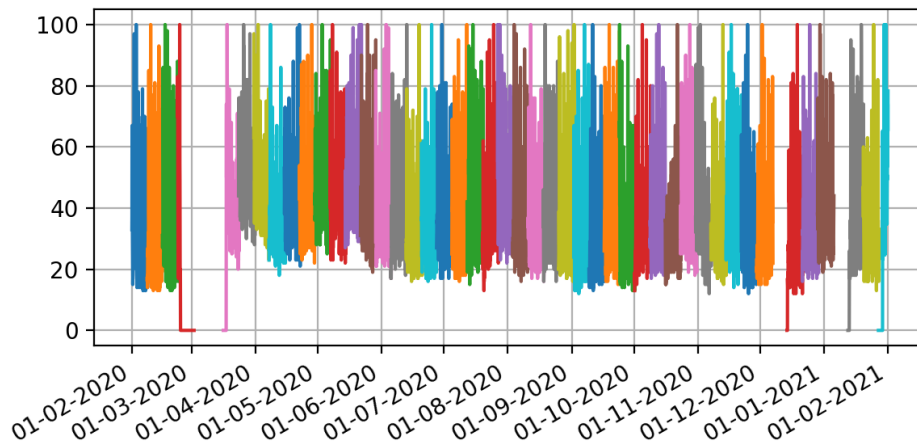


Figure 2.9: Batches of hourly data over a year. Some batches could not be retrieved and some have several values of zero, meaning no data existing for this time and date.

As hourly requests give an interest over time of zero for these intervals, we decided to use daily requests to fill these gaps. Since data collection must yield results which are sampled on a day-by-day basis, we can therefore proceed in several steps once the hourly batches have been collected:

1. Scale the adjacent valid hourly batches, as shown on Figure 2.10. The scaling is done using overlaps as described in section 2.2.4.
2. Drop missing values, such as consecutive zeros at the beginning or end of each batch, and resample the hourly data to daily data, by taking the mean of the values corresponding to the same day. This is represented on Figure 2.11.
3. Use daily requests for deducing the missing values on the gaps. Once this is done, there are interest values over time for each date and the complete signal can be formed.

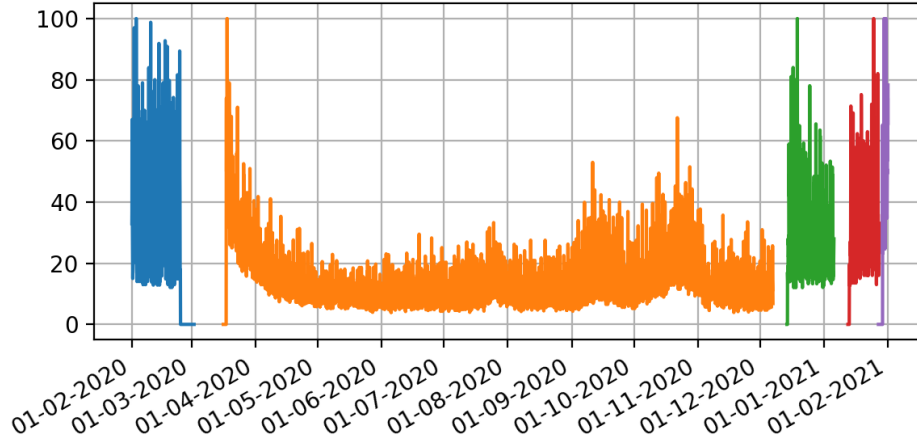


Figure 2.10: Hourly batches merged together, ignoring the neighbors that do not have enough data or are missing

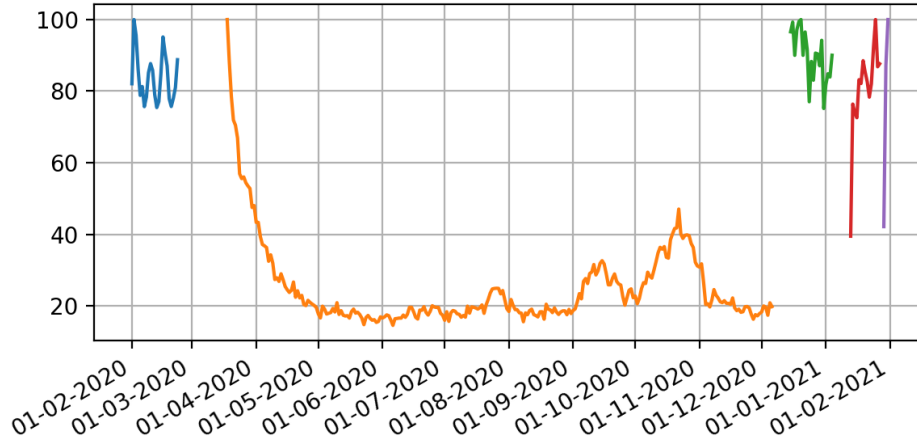


Figure 2.11: Resampling of hourly scaled data to daily data, taking the mean of the values over the same day.

Although this method works, its problem resides in the number of queries that needs to be made. In addition to asking for hourly data, we also need to ask for an average of several daily queries to provide data in the gaps. The number of requests that must be done therefore skyrockets and some connections errors arise, as we cannot simply send several thousands of requests to *Google*'s servers and expect an answer every time. This problem will be covered in the next section.

Search quotas

Google Trends has a limit on how many queries we can make. Indeed, this search quota limits the user to a certain amount of requests per IP address. After this limitation has been reached, an error 429 is thrown and the *Google*'s server stops answering to our queries. The exact limit on the number of queries that can be made per IP address per day is unknown. But, from our experiments, it is not reset every 24 hours, as we

can request less queries if we have reached the limit a few days earlier. As we need to explore as many useful topics as possible, in several countries/regions and over multiple time spans, we need to find a solution that allows us to perform enough queries to collect the data correctly. We have tested 3 different approaches to try and address this problem.

The first solution is to stop the process between two queries for a certain amount of time. This puts less pressure onto *Google*'s servers and the stream of requests is less prone to be considered as an attack. Additionally, we also stop the process for a longer period of time if an error is thrown by *Google*. Each of these waiting periods is characterized by a random time interval of at least one second to prevent regular pattern. This idea was already used by *Derval* et al. in [1].

The second solution is to randomize the order in which the queries are sent. We have observed that *Google Trends* is less prone to ban our IP address when we are using a random order for the queries rather than following the same regular iteration pattern. This can be implemented by preparing the requests without executing them, choosing one of them randomly and execute it. If it succeeds, this requests is removed from the list of requests that needs to be processed. Otherwise it is kept within the list and chosen randomly later on.

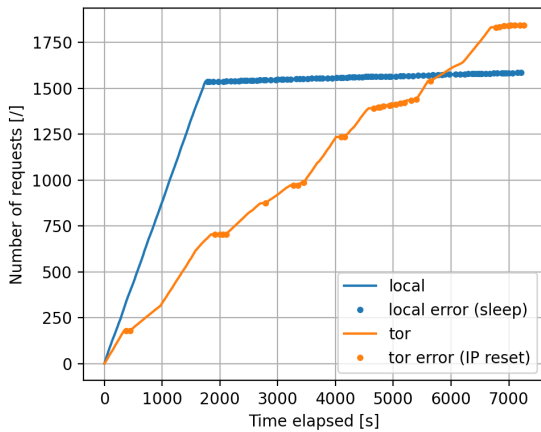


Figure 2.12: Number of requests successfully retrieved from *Google Trends* over time. The dots indicate when an error is encountered

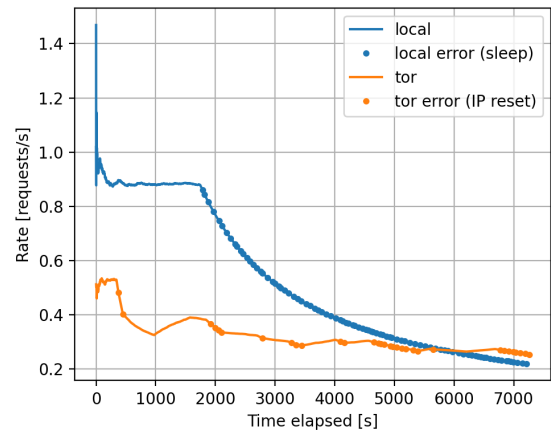


Figure 2.13: Frequency of the requests successfully retrieved from *Google Trends* over time. The dots indicate when an error is encountered

The third solution, giving the most significant results, is to send our requests through a *Tor* proxy [19] instead of using our own local IP address. This requires to install *Tor* to be able to configure it within our program and to send our requests through a proxy rather than by using our local IP address. The requests sent through *Tor* tend to be processed a bit slower. But once the search quota is reached, changing the proxy's IP address allows us to send more queries without being detected by *Google Trends*. Conversely, if we continue to send requests with our local IP address, our requests will simply be constantly blocked by the *Google*'s server. This can be shown on Figures 2.12 and 2.13.

All requests were done using randomly chosen localization, keywords and time spans.

As we observe, once an error is encountered when using our local IP address, only a few queries can be processed thereafter. Indeed, the *Google*'s server detects the IP address that sends the queries and simply blocks it. But when using *Tor*, several changes of IP addresses allows us to query more. A second experiment was done 24 hours after the one that was presented before, whose results are represented on Figures 2.14 and 2.15.

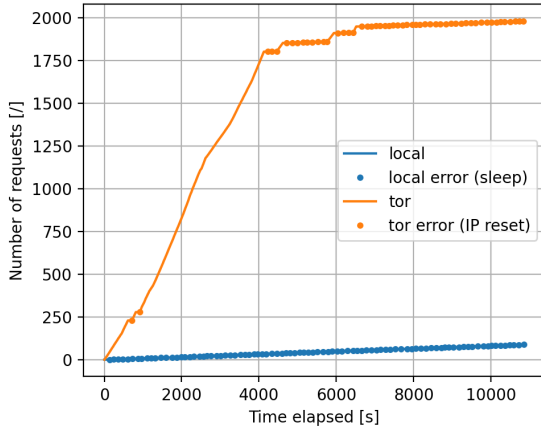


Figure 2.14: Number of requests successfully retrieved from *Google Trends* over time after the first experiments, shown on figures 2.12 and 2.13. The dots indicate when an error is encountered

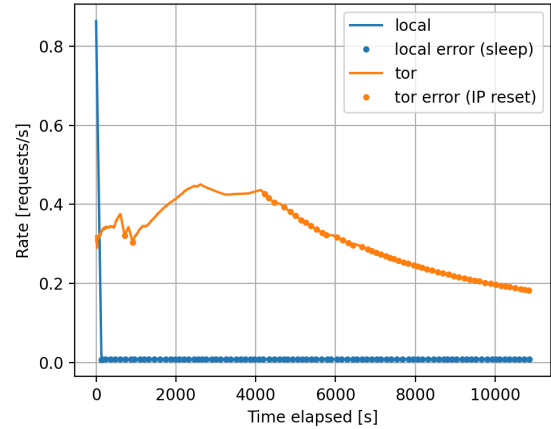


Figure 2.15: Frequency of the requests successfully retrieved from *Google Trends* over time after the first experiments, shown on figures 2.12 and 2.13. The dots indicate when an error is encountered

From this second experiment, we see that *Tor* allows to send much more queries but can also reach a limit. However, it should be possible to prevent this limit by increasing the waiting time used at each stop, i.e. the waiting time between two queries or when an error is returned. Following this experiment, we have changed the waiting time so that it is still chosen randomly but tends to increase depending on the number of errors caught. This should help to fool *Google Trends* even more and be able to send more requests. In the end, to get even better performances, we have combined the three methods explained at the beginning of this section when sending our queries. First, we send our requests using our own local IP address. Once *Google* has sent several exceptions, we switch to *Tor* for the remaining requests. All of those queries are chosen randomly between all the requests that we want to query and a waiting time is applied between each sending.

Low data found on a batch

Some topics are closely related to the coronavirus as they have been recognized as typical symptoms of it. Nevertheless, the interest over time of those topics does not always show values different from zero on *Google Trends*. This means that no data relative to this topic could be found for a particular time interval and localization. For instance, this is the case for "*Agueusia*" in some particular French regions. This is different from

missing batches, and can be observed in both daily and hourly queries.

During our experiments, we have also observed a similar behavior when asking for data over other countries. Some particular topics giving almost only non-zero values in some countries can lead to a few existing values in other ones. Our hypothesis is that the interest over time of a topic is representative of reality if a lot of people use *Google* searches in a particular region and time interval. It means that the more people in the region, the higher the chance of getting non zero values (and realistic results). This is why we can observe few existing values in Luxembourg. Indeed, this country is much less populated than Belgium and the interest over time for most topics therefore contain many zeros.

There is also another factor that can explain the presence of zeros in the interest over time of a particular query: the translation used. It can be problematic since a simple *Google* search in a language cannot always be perfectly linked to the corresponding topic, which should theoretically be valid for all languages. Additionally, because of the cultural differences between people in different countries, there is no guarantee that the same term, even perfectly translated, will be used in the same manner in every country.

Apart from the high number of zeros values, the interest over time of some topics tends to vary a lot from one time stamp to the next. This can be attributed to the same problems as before: there is a low amount of *Google* searches that are performed for this topic in a particular region.

2.2.6 Different collection methods

Using the hourly and daily requests, we ended up with three different methods to collect daily data over a long period of time. Those three methods are :

Hourly collection : where the data is collected using hourly requests. Missing batches are retrieved using a mean of several daily requests.

Daily collection : where the data for a single day is collected using an average of multiple daily requests.

Minimal collection : where the first time the data is collected, daily requests are used and afterwards hourly requests are used. No mean is performed and no gaps are found.

Each of those methods leads to a daily signal for a particular topic and localization. To have a better view at the smoothness of the signal depending on the collection method used, the data collected for the topic *Fever* in Belgium between February the 1st 2020 and February the 1st 2021, is shown on Figures 2.16, 2.17, 2.18. The data given directly by *Google Trends* when using a time span of over a year, and therefore weekly sampled, is represented on Figure 2.19 for comparison.

But the different collection methods have several characteristics that needs to be considered. They are summarized in Table 2.2. Note that the number of requests presented here is considered for a single localization and a single topic.

	Hourly collect	Daily collect	Minimal collect
Request for collecting over a year	$50 + nb_{mean} \times nb_{gap}$	$2 nb_{mean}$	3
Requests for actualizing	1	nb_{mean}	1
Smoothness of the data	smoothest	smooth	not smooth
Latest complete day available	yesterday	3 days before	yesterday

Table 2.2: Comparison between the collection methods. nb_{mean} is the number of daily requests used to form the average of the data and nb_{gap} is the number of time where hourly data could not be retrieved: the number of interval where daily data needs be used in the hourly method. Note that we need to collect data for 50 hourly periods instead of 52, as a hourly request can cover 7 days and 23 hours instead of 7 days.

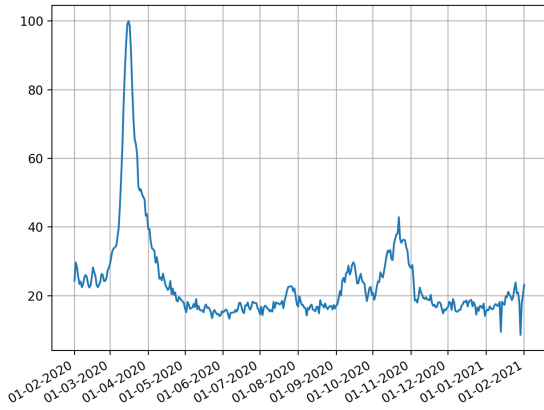


Figure 2.16: Hourly collect for the topic *Fever* on Belgium

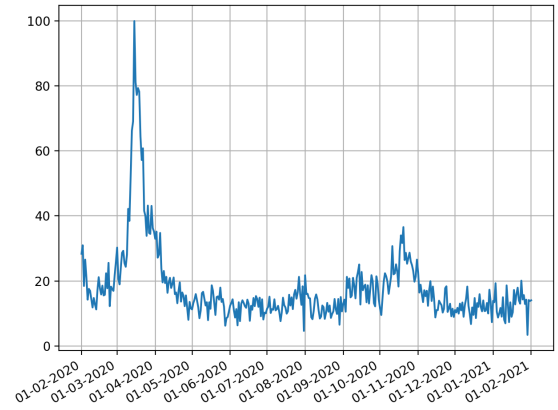


Figure 2.17: Daily collect for the topic *Fever* on Belgium

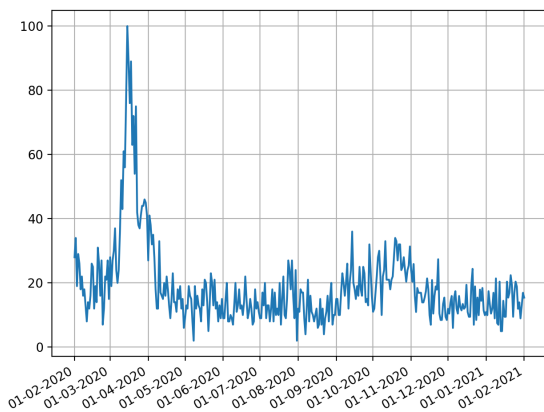


Figure 2.18: Minimal collect for the topic *Fever* on Belgium

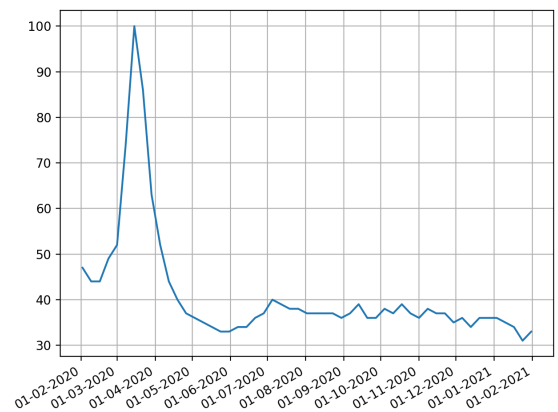


Figure 2.19: Results given *Google Trends* when asked data about *Fever* in Belgium for a year. The data is weekly sampled as it exceeds 9 months

Because we wanted to consider as many topics as possible, have the latest data available and be able to do a lot of requests without being banned by *Google Trends*, we have chosen to use the minimal collect method for collecting data relative to a new topic, i.e. a topic that has not yet been collected. However, we will continue to collect data on an hourly basis for the topics already collected since this method is the least noisy.

The objective of the data preparation stage is to put the data in the best possible format to be used in a prediction model later on. This includes three different steps that will be described in the rest of this chapter: data preprocessing, data transformation and data augmentation.

In computer science, the *garbage in, garbage out* (GIGO) concept is well-known: nonsense input data produces nonsense output [20]. It is thus easy to understand that using good data is more important than creating complex prediction models. Therefore, data preparation is an extremely important step and can never be bypassed.

3.1 Data preprocessing

The real-world data tend to be incomplete, inconsistent and noisy. The quality of the collected data is then poor and this situation can lead to poor behavior of the models that will be built from that data. Therefore, the goal of data preprocessing is to transform the raw data into clean data that can be used to interpret and understand links between features. Note that the data that has been collected is stored in a Pandas dataframe [21]. A dataframe is a two dimensional tabular data with labelled rows and columns. Table 3.1 shows an example of such a dataframe. We observe that the different features are present in the columns of the dataframe: the number of total and new hospitalizations as well as the interest over time for three *Trends* related to the coronavirus.

Note that in the dataframe shown in Table 3.1, the column containing the number of new hospitalizations is the target columns, i.e. the one that we will try to predict in the future. Other columns are input data that will be given to the prediction model to help it predict the number of new hospital admissions. Nevertheless, all these columns should be preprocessed before being usable.

LOC	DATE	TOT_HOSP	NEW_HOSP	Fever	Sore throat	Symptom
BE	2021-04-16	3015	-17.0	20.51	22.07	22.82
	2021-04-17	2910	-105.0	22.29	23.82	21.99
	2021-04-18	3000	90.0	20.41	24.29	24.29

Table 3.1: Example of a part of a dataframe that contains the collected data for Belgium

3.1.1 Rolling average

A very important preprocessing method we need to apply is a centered rolling average on each column of the dataframe containing the collected data. This method is used in order to smooth the raw data we got (which can be very unstable). The principle is the following: each element of the column becomes the average of the `window` preceding days. If the index of the element is strictly lower than `window`, a `NaN` value is set to the element. For example, in Table 3.2, we observe that a rolling average of 3 is performed on the initial vector. Therefore, the two first elements of the resulting vector are `NaN` values. The third element of this vector is equal to the average of the three first elements of the initial vector : $\frac{10+20+40}{3} = 23.3$. The rest of the final vector is computed in the same way.

Initial vector	10	20	40	50	80	10	90	40	20	60
After rolling average of 3	NaN	NaN	23.3	36.7	56.7	46.7	60	46.7	50	40

Table 3.2: Diagram of the use of rolling average

In our case, we decided to use a rolling average with a `window` value of 7 days. This can be justified by the fact that some data might not be taken into account at weekends and may be counted on another day of the week to compensate for this. As a result, instabilities occur in the data and the rolling average of one week helps to smooth these instabilities.

3.1.2 Removing NaN values

If some values are missing or undefined in the dataframe, they are replaced by a `NaN` value. Unfortunately, the prediction model is not able to handle these values as they do not represent a number. Therefore, we have to remove these values during the preprocessing phase.

Since some `NaN` values were introduced because of the rolling average, the next preprocessing step is to remove these `NaN` values from the dataframe. These values may also be due to different data collection in some regions. Indeed, in some regions we have access to 2-day old data while in other regions we only have access to 3-day old data. Therefore, we need to add a `NaN` value to the end of the data vector for the first region in order to store all the collected data in the same dataframe. Remember that it is impossible to store 2 columns of different sizes in the same dataframe.

3.1.3 Feature scaling

The features of the dataframe containing the data are not all at the same scale. Indeed, the data collected through *Google Trends* are scaled between 0 and 100 while the total number of hospitalizations is a positive number that has no upper bound. For the number of new hospitalizations, the scale is different again as the values can be positive or negative and are not bounded. Therefore, the last preprocessing step that we need to apply is a feature scaling of our data. To do this, we have used two different well-known

methods.

On the one hand, we used a normalization of our data between 0 and 1. The goal of such a normalization is to change the values of all numerical columns in the dataframe to a standard scale, without distorting the variations in the ranges of values. Therefore, the impact of outliers will still be present after the scaling. The geometrical intuition behind this transformation is that the n-dimensional data is squashed into a unitary n-dimensional hypercube [22]. This scaling is useful when we do not know which distribution does the data follow. Formally, the formula that is used to normalize the dataframe values is as follows :

$$X_{new} = \frac{X - X_{min}}{X_{max} - X_{min}} \quad (3.1)$$

where :

- X is the current element in the dataframe
- X_{min} is the minimum value of the dataframe
- X_{max} is the maximum value of the dataframe
- X_{new} is the new element that will replace the current element in the dataframe

On the other hand, we used a standardization of our data. This method can be helpful if data follow a Gaussian distribution. The goal is to ensure a zero mean and a unit standard deviation in the data. Therefore, the initial shape of the distribution is not affected by the scaling but the impact of outliers is reduced. Unlike normalization, the data will not be bounded after the scaling. Formally, the formula that is used to standardize the dataframe values is as follows :

$$X_{new} = \frac{X - \mu}{\sigma} \quad (3.2)$$

where :

- X is the current element in the dataframe
- μ is the mean value of the dataframe
- σ is the standard deviation value of the dataframe
- X_{new} is the new element that will replace the current element in the dataframe

Using those scaling, we will predict values in the scaled range and then convert it back to the initial range. This also comes handy when using machine learning techniques, as we will describe in chapter 4.

3.2 Data transformation

The purpose of data transformation is, as the name suggests, to transform data from one format to another. The transformed data may be easier to use and understand for both humans and computers. For example, data transformation can help a prediction model to understand the links between features more easily. The transformations can be applied to every features of the dataframe.

Before we start applying transformations to the cleaned data, let us first analyze these data. Figure 3.1 shows the evolution of both the total number of hospitalizations (a) and the number of new hospitalizations (b) in Belgium. We are clearly observing the two waves we have experienced in 2020 in Belgium. Remember that the total number of hospitalizations at a particular time represents the total number of people who are in the hospital at that time. This value is always greater or equal to 0. The interpretation of the number of new hospitalizations is a little bit more complex to understand. Indeed, it represents the number of people who entered the hospital on a certain date minus those who left. In other words, the number of new hospitalizations that we consider can be defined as the difference between the number of admissions and the number of hospital discharges. That is why this value can be either positive or negative.

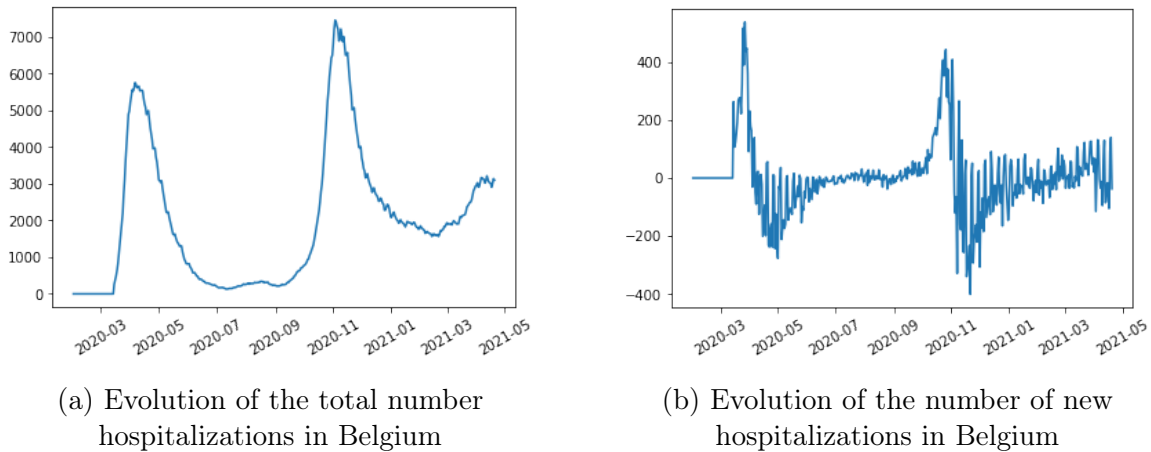
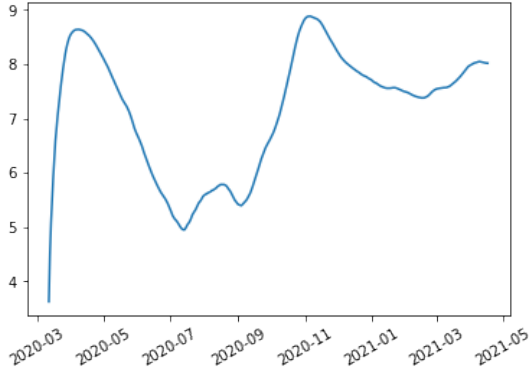


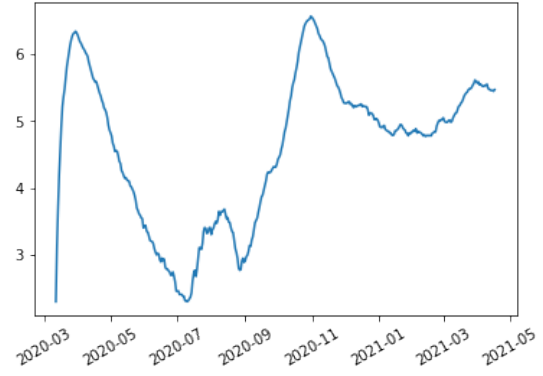
Figure 3.1: Initial data before doing transformations

3.2.1 Logarithmic transformation

The logarithmic transformation is a widely used method for handling skewed data. It is useful in order to give more importance to small values, i.e. values close to 0 and to reduce the importance of outliers. Indeed, when the original data are skewed, we can log-transform them to make them approximately conform to normality. This is important for the results of the statistical analysis that are done on these data to be valid [23]. Figure 3.2 shows the evolution of both the total (a) and the new (b) number of hospitalizations in Belgium after the transformation.



(a) Evolution of the logarithmic total number of hospitalization in Belgium

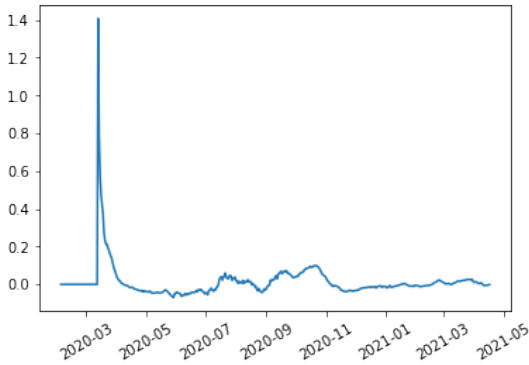


(b) Evolution of the logarithmic number of new hospitalization in Belgium

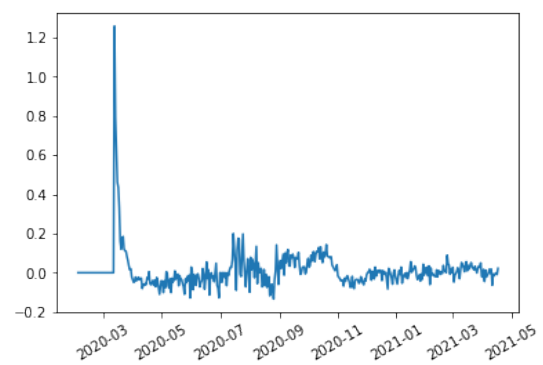
Figure 3.2: Data after the logarithmic transformation

3.2.2 Percentage change transformation

The percentage change makes it possible to quantify the change in a numerical quantity between two dates. In our case, we try to quantify the change in number of hospitalizations between two consecutive dates. This transformation method could be useful to have percentage data as input of our model for one main reason. The prediction model is not able to make a consistent prediction when it has to predict a value it has never encountered. With this transformation, the model will only have to predict values that it has met before, which will improve the quality of its predictions. Figure 3.3 shows the evolution of both the total (a) and the new (b) number of hospitalizations in Belgium after the transformation.



(a) Evolution of the percentage change in the total number of hospitalization in Belgium



(b) Evolution of the percentage change in the number of new hospitalization in Belgium

Figure 3.3: Data after the percentage change transformation

3.2.3 Shifting

During our analysis of *Google Trends* data, we noticed that the interest over time of some topics are highly correlated with the total number of hospitalizations but with a delay of a few days. Therefore, we decided to create $n_samples - 1$ new columns for each

feature in the dataframe that will be given as input of the prediction model. These new columns represent the interest over time of each topic lagged by a certain number of days. For a better view, you can have a look at Table 3.3. We observe that the initial *Google Trends* data for the topic "Fever" are in the column named "Fever". Nevertheless, other columns were also added. We observe that the column "Fever(t-1)" is composed of the same data as the column "Fever" but with a lag of one day. The same process is observed for all features present in the dataframe.

NEW_HOSP(t-2)	Fever(t-2)	NEW_HOSP(t-1)	Fever(t-1)	NEW_HOSP	Fever
0.339	0.067	0.342	0.061	0.347	0.071
0.342	0.061	0.347	0.071	0.340	0.085
0.347	0.071	0.340	0.085	0.329	0.084

Table 3.3: Example of a lagged dataframe for the topic "Fever" in Belgium

On Figure 3.4, we observe the evolution of the data relative to symptoms of the coronavirus collected through the *Google Trends* tool as well as the evolution of the **total number of hospitalizations** in Belgium over time. We note that there is a link between these two types of data. Indeed, we note that the curve representing the total number of hospitalizations and the curves representing the *Google Trends* data have a similar shape even if the amplitude of the curves is different. We also observe that the curves of the Trends data do not reach their maximum at the same time as the hospitalizations curve. In fact, the *Trends* data curve seems to be a few days ahead of the hospitalization data. These curves are therefore more correlated with a certain number of days of lag. In particular, we note that the curve of the symptom "Anosmia" is extremely similar to the hospitalizations curve. This can be explained by the fact that this symptom is a typical symptom of a coronavirus infection.

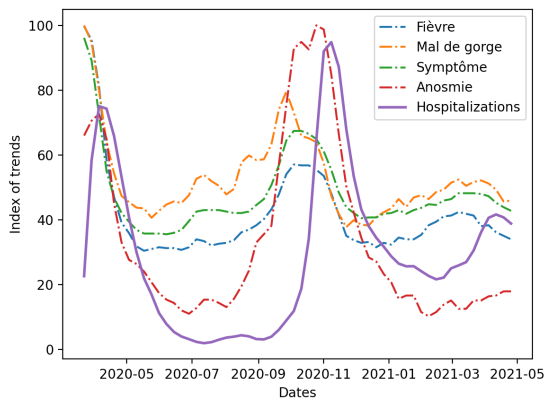


Figure 3.4: Total hospitalizations and *Trends* relative to some symptoms of the coronavirus in Belgium

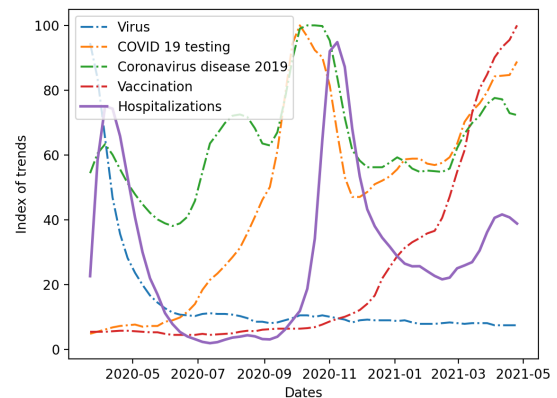


Figure 3.5: Total hospitalizations and *Trends* relative to the coronavirus in general in Belgium

On figure 3.5, we observe the evolution of the data relative to the coronavirus in gen-

eral as well as the evolution of the **total number of hospitalizations** in Belgium over time. Contrary to the curves representing the symptoms, we observe here that not all curves representing the topics have a similar shape as the one of the hospitalizations. Indeed, the topic "Virus" is correlated with the first wave of the coronavirus but does not seem to have been researched much afterwards. On the contrary, the topic "Vaccination" seems to be very much researched in recent months but this was not the case before. This can be explained by the fact that the vaccination campaign started a few months ago in Belgium. Finally, the topic "COVID 19 testing" seems to be highly correlated with the second wave of the coronavirus. This can be explained by the fact that the number of tests relative to the coronavirus has increased exponentially from October 2020.

On Figure 3.6, we observe that the blue curve, i.e. the one representing the evolution of the topic "Fever" in function of the number of days of lag, seems to be the one which is the most correlated with the curve representing the total number of hospitalizations in Belgium but with 35 days of lag. The topics "Symptom" and "Anosmia" seem to reach a maximal correlation with a shift of 27 days. Concerning the topic "Sore throat", the maximal correlation seems to be reached with 47 days of lag.

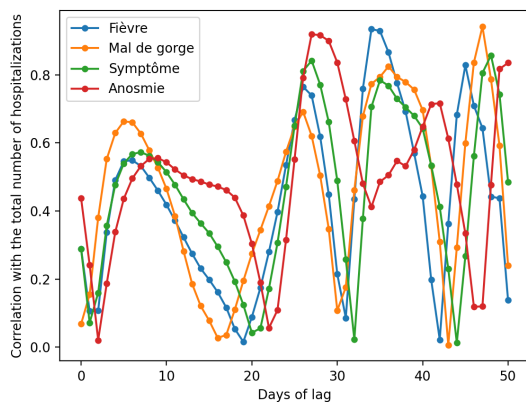


Figure 3.6: Correlation between *Trends* relative to symptoms of the coronavirus and the total number of hospitalizations in Belgium with respect to the number of days of lag

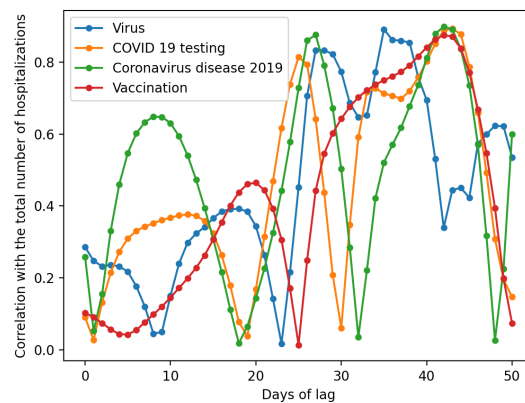


Figure 3.7: Correlation between *Trends* relative to the coronavirus in general and the total number of hospitalizations in Belgium with respect to the number of days of lag

On Figure 3.7, it is easy to observe that three different topics reach their maximum correlation at the same time. Indeed, the correlation of the topics "Vaccination", "Coronavirus disease 2019" and "COVID 19 testing" is maximal when the number of days of lag is equal to 42. Concerning the topic "Virus", the maximum seems to be reached a little bit before that since the number of lag that leads to the highest correlation is equal to 37.

3.3 Data augmentation

In order to predict the evolution of the number of hospitalizations, we decided to use models that are at the cutting edge of technology at the moment, neural networks [24]. However, this type of model requires a large amount of data to give satisfactory results. As the current global pandemic was only a few weeks old when we started developing the prediction model, we had to find a solution to the lack of data. That is why we started to use data augmentation methods.

In data analysis, data augmentation refers to techniques that are used to increase the amount of data by adding slightly modified copies of already existing data or by creating new synthetic data from existing data [25]. In our case, we have created new synthetic data from already existing data.

3.3.1 For France/Belgium

Before applying any data augmentation method, we had *Google Trends* data for Belgium and for 22 different French regions. On each of these regions, we have collected data for 12 different topics. Nevertheless, all these topics were not usable on all regions. Indeed, some regions showed completely inconsistent results that cannot be used for the prediction. This may be due to the fact that the region is not sufficiently populated and therefore few *Google* searches were performed as discussed in section 2.2.5. This leads to a similar graph as the one represented on Figure 3.8. Note that this graph has already been reworked to try to recover as much information as possible but despite the efforts made, the data remains very unstable.

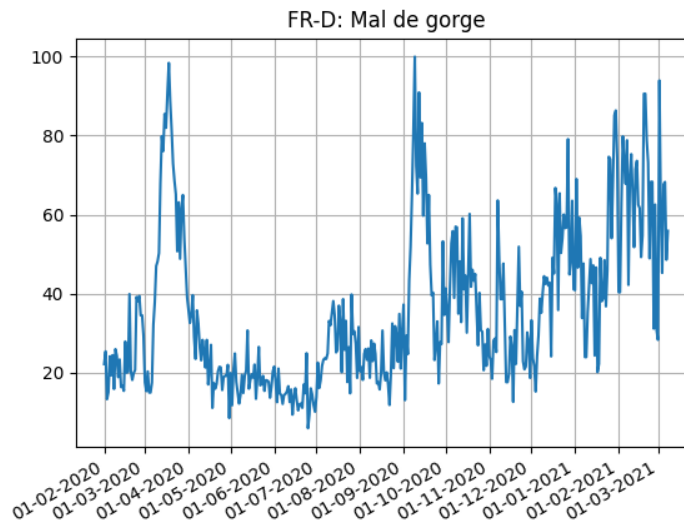


Figure 3.8: Evolution of the topic *Mal de gorge* on the FR-D region

The solution we decided to use in order to generate more data is to merge several adjacent French regions together. Thanks to this method, we avoid the problem of lack of

population in some regions since we add up the populations of several regions. We have merged the French regions in several ways : we merged the regions 2 by 2 and then 2 by 2 and 3 by 3 and so on until a sufficient amount of data is available to train a model properly.

Thanks to this method, we managed to obtain a total of 172 different French regions instead of 23. We could have had more by assembling regions randomly. However, for the sake of realism, we have decided to assemble only regions that are side by side. That means that we could merge regions FR-F and FR-D to create a new region but it was not allowed to merge regions FR-F and FR-U. Figure 3.9 shows the initial French regions that we had before applying data augmentation methods.

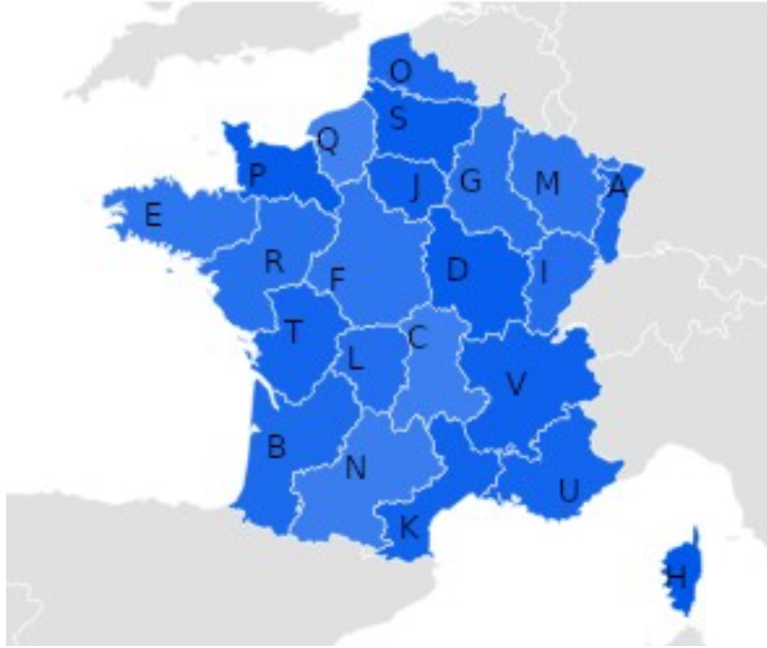


Figure 3.9: Initial French regions before merge. The letter corresponds to the codes used by *Google Trends* for a given region: FR-A, FR-B, ...

Now that we have merged the French regions to increase the number of data, we have to find the new *Google Trends* data and the number of hospitalizations for these new regions. Indeed, *Google Trends* does not allow us to combine certain regions. As far as hospitalizations are concerned, they are reported by region in the official document we have collected as explained in section 2.1.2. Therefore, we decided to apply two simple rules for every augmented regions :

- The number of hospitalizations is equal to the sum of the number of hospitalizations of the regions forming it.
- The *Google Trends* data are equal to the *Google Trends* data weighted by the population of the regions forming it.

3.3.2 For Europe

Despite data augmentation, some French regions still have a lack of stability. Moreover, we also wanted to expand our dataset beyond Belgium and France. This is why we decided

to take a different approach. We collected data from different European countries and then augmented these data in the same way as the French data. The only difference is that we work with data on countries instead of regions. After having applied the data augmentation, we have 82 different European regions at our disposal.

In recent times, the field of machine learning and deep learning has seen a tremendous resurgence of interest as new methods have emerged. Inspired by the human brain, these new models can perform much better than previous forms of artificial intelligence. In order to better understand these methods, let us go back to the basis of artificial intelligence.

4.1 Artificial Neural Network

The first step towards artificial neural networks was taken in 1943 when *Warren McCulloch* and *Walter Pitts*, a neurophysiologist and a young mathematician respectively, wrote a paper on how neurons might work. They created a simple computational model for neural network with electrical circuits [26].

Neural networks are a metaphor for brain structures: assemblages of elementary constituents, each of which performs a simple treatment but whose single local interest brings out global properties worthy of interest. Each component operates independently of the others, so that the whole is a highly interconnected parallel system. The information held by the neural network is distributed throughout the set of constituents and not localized in a part of memory. Finally, a neural network does not program itself to carry out this or that task. It is trained on acquired data, thanks to a learning mechanism that acts on the network's components in order to perform the desired task as well as possible [27].

4.1.1 Components of an Artificial Neural Network

All neural networks contain layers that are themselves filled with nodes called neurons. Each neuron of a layer is connected to all neurons of the next layer. Each of these connections is associated to a weight that represents its strength. When the network receives an input at a given neuron in the input layer, this input is transmitted to the next layer thanks to the connection and will be multiplied by the weight assigned to it [27]. For each node in the second layer, a weighted sum is computed, taking into account all the incoming connections. This sum is then given to an activation function, which performs some linear or nonlinear transformation on the given sum. Formally, a neuron can be defined as :

$$Y = \sum(input * weight) + bias \quad (4.1)$$

Layers

The simplest models are composed of a single layer and are used in order to represent linearly separable functions. This means that the problem that we try to solve is a binary classification problem where the two classes can be separated by a line. Nevertheless, most problems that we are interested in solving are not linearly separable. That is why we need to build more complex models. There are three main types of layer :

- **Input layer** : contains the initial data given as input of the model
- **Hidden layer** : is located between the input and the output layers of the model. These layers perform nonlinear transformation of the inputs entered into the network.
- **Output layer** : contains the output of the model

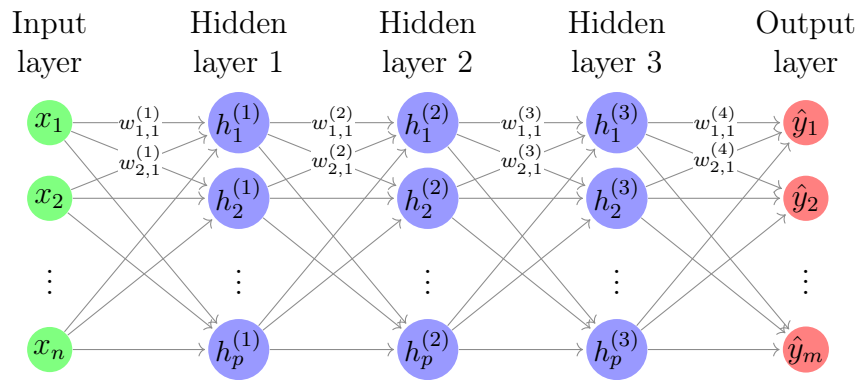


Figure 4.1: Representation of an artificial neural network

On Figure 4.1, we observe an artificial neural network composed of an input layer, an output layer and three hidden layers. Each of these layers can include a different number of neurons. Since this model has more than two hidden layers, it can be considered as a deep learning model.

Activation function

The choice of activation function to be applied in the hidden layers will control the ability of the neural network to learn the training set. In the output layer, the activation function that is applied will define the type of predictions the model will make. For the hidden layers, there are mostly three different main activation functions that can be considered :

- **Rectified Linear Units (ReLU)** : This function looks like a linear function. Indeed, it will output the input directly if it is positive and it will output zero otherwise. Formally, the activation is given by $A = \max(0, x)$ where x represents the input received by the neuron. The output domain of this function is included between 0 and $+\infty$.

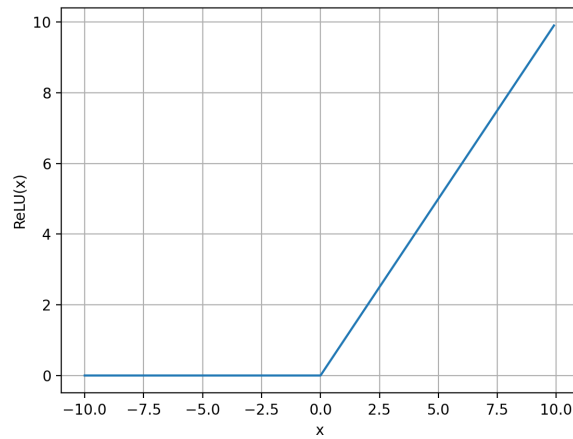


Figure 4.2: Representation of ReLU activation function

- **Logistic/Sigmoid** : This activation function is nonlinear. Formally, the activation is given by $A = \frac{1}{1+e^{-x}}$ where x represents the input received by the neuron. The output domain of this function is included between 0 and 1.

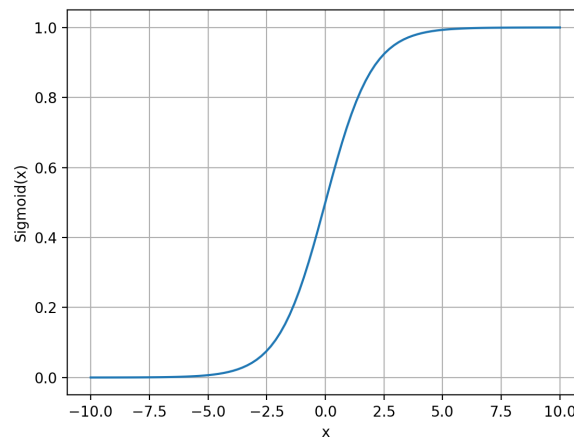


Figure 4.3: Representation of sigmoid activation function

- **Hyperbolic tangent (tanh)** : This activation function is very similar to the sigmoid one except that the output domain is not the same. Formally, the activation is given by $A = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$ where x represents the input of the neuron. The output domain of this function is included between -1 and 1.

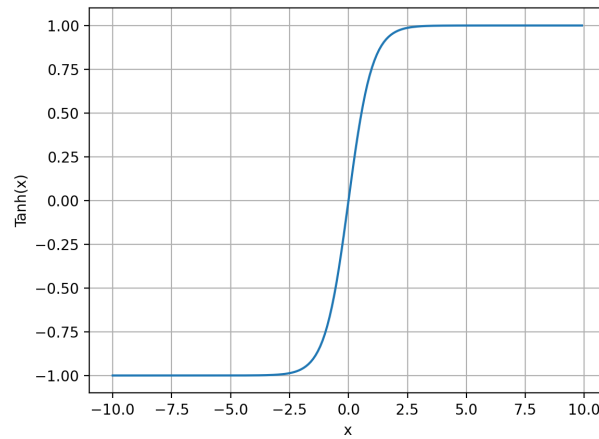


Figure 4.4: Representation of hyperbolic tangent activation function

Among those activation functions, the ReLU activation function is the most commonly used in hidden layers. Figure 4.5 shows which activation function is often used with different type of neural networks.

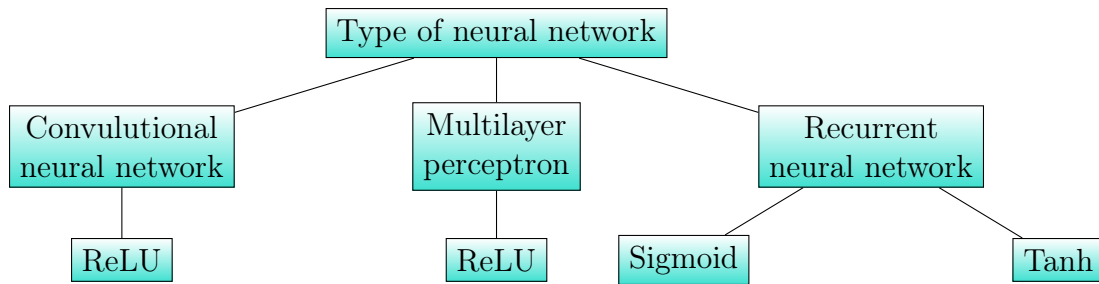


Figure 4.5: Activation function used based on the type of neural network

Now that we have an idea of which activation function can be used in the hidden layers, let us have a look at the most commonly activation functions for the output layer.

- **Linear** : It multiplies the inputs by the weights related to each neuron before creating an output signal proportional to the input. Formally, the activation is given by $A = Cx$ where C is a constant and x represents the input received by the neuron. This function has no bound and therefore, the output of it can be from $-\infty$ to $+\infty$.

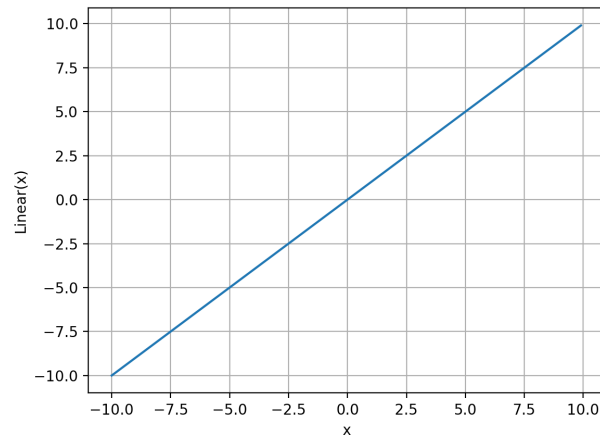


Figure 4.6: Representation of linear activation function (C=1)

- **Logistic/Sigmoid** : This activation function was defined previously.
- **Softmax** : This function will output only one value for each neuron of the output layer. The output domain of this function is included between 0 and 1 but the total sum of each output neuron should be equal to 1. Therefore, the output of this activation function is a probability distribution.

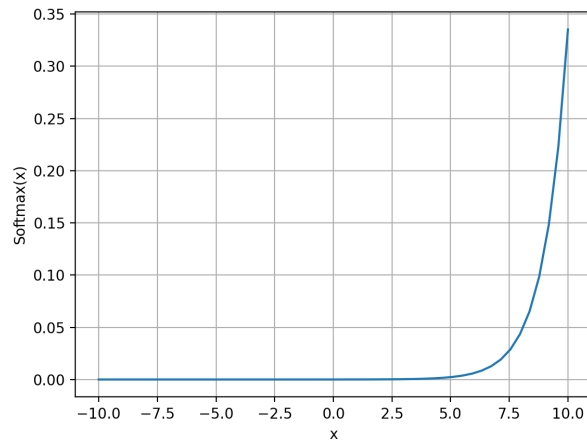


Figure 4.7: Representation of softmax activation function

Among those activation functions, the sigmoid activation function is the most commonly used in the output layer. Figure 4.8 shows which activation function is often used depending on the problem being faced.

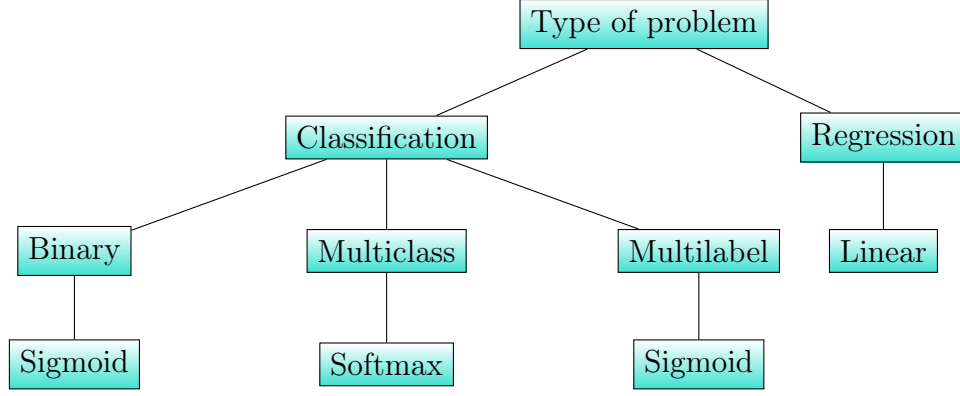


Figure 4.8: Activation function used based on the type of problem

4.1.2 Learning

The learning phase of a neural network consists of adapting the network to better handle a specific task by considering the training data. Every neural network learns by modifying its weights in order to improve the accuracy of the model [28].

Loss/Cost function

In the context of optimization, the objective function is the function used to evaluate a candidate solution, i.e. a set of weights for each neuron. In the context of neural networks, the goal is to minimize the error and the objective function is referred to a loss or a cost function. There are a lot of different available functions but the most commonly used are:

- **Mean Squared Error (MSE)** : This function is computed as being the average of the squared differences between the predicted and actual values. Formally, this function is defined as :

$$L = \frac{1}{n} \sum_{i=1}^n (\hat{y} - y)^2 \quad (4.2)$$

where n is the number of outputs of the network.

- **Mean Absolute Error (MAE)** : This function is computed as being the average of the absolute differences between the predicted and actual values. Formally, this function is defined as :

$$L = \frac{1}{n} \sum_{i=1}^n |\hat{y} - y| \quad (4.3)$$

where n is the number of outputs of the network.

- **Root Mean Square error (RMSE)** : This function is an extension of the MSE. It is computed as the average of square root of sum of squared differences between predictions and actual values. Formally, this function is defined as :

$$L = \sqrt{\frac{1}{n} \sum_{i=1}^n (\hat{y} - y)^2} \quad (4.4)$$

where n is the number of outputs of the network.

Backpropagation

Backpropagation is the algorithm charged to update the weights and biases of a neural network. Its goal is to be able to get values for the parameters of the network such that the cost function is minimized. This algorithm computes iteratively the values Δw to add to the weights w and Δb to add to the biases b . It begins by first updating the weights of the last layer before iterating over the layers before him (hence the 'back' propagation). On the last layer, the outcome of the model is defined as

$$\hat{y}_j = a_j^{(L)} = \sigma \left(z_j^{(L)} \right) \quad (4.5)$$

Where the superscript $^{(L)}$ indicates which layer the term is related to, the layer $^{(L)}$ being the last one, $a_j^{(L)}$ is the activation function of the neuron j and $z_j^{(L)}$ is the input given to the function σ to compute the activation. This term is computed as the weighted sum of the activation functions of the preceding layers, plus a bias:

$$z_j^{(L)} = w_{j,0}^{(L)} a_0^{(L-1)} + w_{j,1}^{(L)} a_1^{(L-1)} + \dots + w_{j,n-1}^{(L)} a_{n-1}^{(L-1)} + b_j^{(L)} \quad (4.6)$$

Those terms are illustrated on Figure 4.9

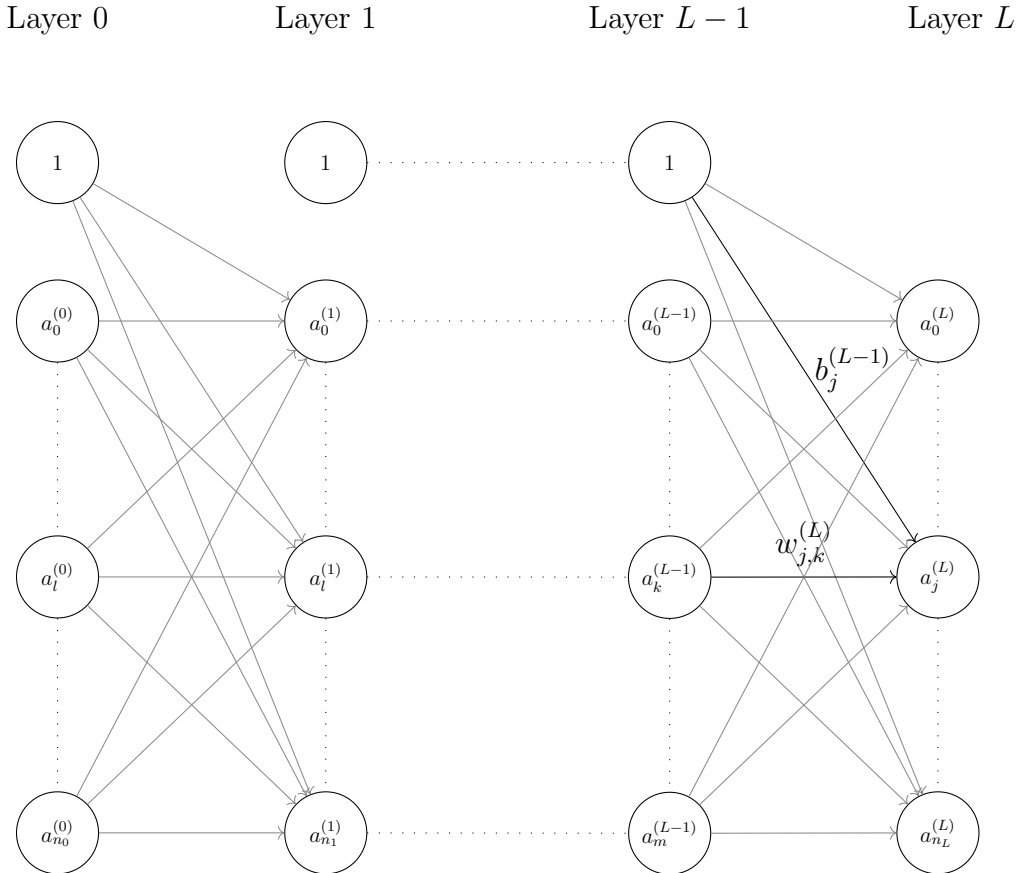


Figure 4.9: Weights, biases and activations functions of the network

Giving this set of numbers, the backpropagation wants to minimize the cost. Therefore, it needs to compute the impact of each of those terms to the cost function C . The cost function is computed term by term, we will denote C_0 as the cost related to the first training example. We first begin by computing the impact of the weights to the cost function of this example: $\frac{\partial C_0}{\partial w_{j,k}^{(L)}}$. Using the chain rule, we can develop this term as

$$\frac{\partial C_0}{\partial w_{j,k}^{(L)}} = \frac{\partial z_j^{(L)}}{\partial w_{j,k}^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}} \quad (4.7)$$

Each of these derivatives can be computed. To be more specific, when using the square error as the cost function, the loss relative to the first training example is expressed as:

$$C_0 = \sum_{j=0}^{n_L-1} \left(a_j^{(L)} - y_j \right)^2 \quad (4.8)$$

From which we directly obtain our first derivative:

$$\frac{\partial C_0}{\partial a_j^{(L)}} = 2 \left(a_j^{(L)} - y_j \right) \quad (4.9)$$

Additionally, the other derivatives can be computed through equations 4.6 and 4.5:

$$\frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} = \sigma' \left(z_j^{(L)} \right) \quad (4.10)$$

$$\frac{\partial z_j^{(L)}}{\partial w_{j,k}^{(L)}} = a_k^{(L-1)} \quad (4.11)$$

However, this only gives the impact that the weights $w_{j,k}^{(L)}$ have on the cost function of the first example, C_0 . The sensitivity to the bias is also found using the chain rules, which let us reuse some derivatives that were computed before:

$$\frac{\partial C_0}{\partial b_j^{(L)}} = \frac{\partial z_j^{(L)}}{\partial b_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}} \quad (4.12)$$

$$= 1 \times \sigma' \left(z_j^{(L)} \right) \times 2 \left(a_j^{(L)} - y \right) \quad (4.13)$$

Finally, the impact of the activation function of the previous layer to the cost function can also be computed. In contrary to the weights and the biases, the activation function $a_k^{(L-1)}$ influences all activations functions of the next layers instead of a single one, $a_j^{(L)} \quad \forall j \in 0, \dots, n_L - 1$. We therefore get:

$$\frac{\partial C_0}{\partial a_k^{(L-1)}} = \sum_{j=0}^{n_L-1} \frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial C_0}{\partial a_j^{(L)}} \quad (4.14)$$

Where the sensitivity of the function $z_j^{(L)}$ with respect to the activation function of

the previous layer is retrieved from equation 4.6:

$$\frac{\partial z_j^{(L)}}{\partial a_k^{(L-1)}} = w_{j,k}^{(L)} \quad (4.15)$$

Combining the previous equations and generalizing for all layers, we get:

$$\frac{\partial C_0}{\partial w_{j,k}^{(l)}} = a_k^{(l-1)} \sigma' \left(z_j^{(l)} \right) \frac{\partial C_0}{\partial a_j^{(l)}} \quad (4.16)$$

$$\frac{\partial C_0}{\partial b_j^{(l)}} = \sigma' \left(z_j^{(l)} \right) \frac{\partial C_0}{\partial a_j^{(l)}} \quad (4.17)$$

Where $\frac{\partial C_0}{\partial a_j^{(l)}}$ is found by combining the equations 4.9, 4.10, 4.14 and 4.15 :

$$\frac{\partial C_0}{\partial a_j^{(l)}} = \begin{cases} 2(a_j^{(l)} - y_j) & \text{if } l = L \\ \sum_{k=0}^{n_{l+1}-1} w_{j,k}^{(l+1)} \sigma' \left(z_j^{(l+1)} \right) \frac{\partial C_0}{\partial a_j^{(l+1)}}, & \text{otherwise} \end{cases} \quad (4.18)$$

This computation is done for every training example and gives the gradient $-\Delta C$, which gives the direction minimizing the cost. Once it is computed, it is combined with a learning rate $\eta > 0$. This learning rate controls how much the model should be modified in response to the estimated error each time the weights of the model are updated [28]. The amounts $\Delta w_{j,i}^{(l)}$ added to the weights $w_{j,i}^{(l)}$ are given by the following formula

$$\Delta w_{j,i}^{(l)} = -\eta \frac{\partial C}{\partial w_{j,i}^{(l)}} \quad (4.19)$$

Similarly, the amounts $\Delta b_j^{(l)}$ added to the biases $b_j^{(l)}$ are given by

$$\Delta b_j^{(l)} = -\eta \frac{\partial C}{\partial b_j^{(l)}} \quad (4.20)$$

The computation of this gradient is relatively expensive, because the cost function needs to be computed for each training example. To reduce it, several algorithms have been created, for instance the stochastic gradient descent. It assumes that the actual cost function can be estimated by the cost function of a mini batch of training examples instead of the cost function of the whole data. This gives a significant speedup when training a neural network. More details are described in the next sub section, about the optimizers.

Optimizers

The learning rate is a hyper-parameter that takes a small positive value. It is given as a parameter of the optimizer, i.e. the function that is in charge of reducing the loss. The learning rate controls how much the model should be modified in response to the estimated error each time the weights of the model are updated [28]. There are several different optimization algorithms that can be used to avoid computing the whole cost function. The most commonly used are :

- **Adam:** It is an extension to the stochastic gradient descent algorithm. This optimization algorithm is based on adaptive estimation of first-order and second-order moments.
- **RMSProp:** This optimization algorithm is similar to the gradient descent algorithm with momentum. The difference between the two methods resides essentially in the calculation of the gradient.

Vanishing gradient problem

A problem that can be encountered when using machine learning techniques is known as the vanishing gradient problem [29]. During the computation of the gradient, the computed terms can become so small that they prevent the weights from changing their values. Even worse, it can completely stop the learning process in some cases.

This can be the case for instance with some activation function outputting a result between 0 and 1, such as the hyperbolic tangent or the logistic function described earlier. Multiplying several times those small numbers can decrease a lot the values through layers. ReLU, on the other hand, can output larger values and suffer less from this problem.

4.2 Architecture of a Recurrent Neural Network

A RNN can be seen as a type of ANN that is specialized in processing sequences. RNNs are often used for dealing with natural language processing or time-series prediction tasks because it contains a memory state. Indeed, dealing with human language requires an understanding of context and the model must therefore be able to retain the past context of a sequence. For time series predictions, the analysis is similar. To be able to make a correct temporal prediction, the model must be able to retain past events.

More formally, a RNN is designed to give good performance in supervised learning where input data are sequential. The main component of this network is that it has a memory state. It can remember information from the past, which is obviously useful for predicting targets that are time-dependent. Figure 4.10 represents the architecture of a RNN. The most important variables of such a network are highlighted :

- x_t : the input of the network at time t ;
- $\hat{y}_t$: the output of the network at time t ;
- h_t : the state at time t contained in the hidden layer.

In this network, we have a recurrence since the hidden state at time t is given to the next state, at time $t + 1$. Therefore, it is a discrete process where at any given time step, the hidden state is computed based on both the previous hidden state and the input data. In order to be able to observe this process through time, Figure 4.11 represents the flow of data at different time steps

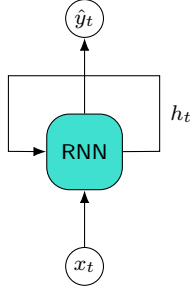


Figure 4.10: Behavior of a RNN

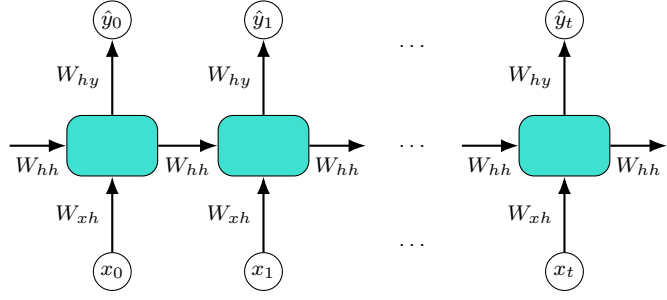


Figure 4.11: RNN through time

This RNN can be described mathematically thanks to two equations :

$$\begin{aligned} h_t &= \tanh(W_{hh}^T h_{t-1} + W_{xh}^T x_t) \\ \hat{y}_t &= W_{hy}^T h_t \end{aligned} \quad (4.21)$$

The parameters W_{hh} , W_{xh} and W_{hy} are the weights of the RNN, and are shown on Figure 4.11. They can be learned by unfolding the recurrence through time, this is called back-propagation through time. In this process, the same weights are used at each time step and only x_t , h_t and y_t evolve with respect to time. However, networks that use back-propagation through time have a hard time to converge for long term dependencies since it can lead to a problem of vanishing gradient. Indeed, in such methods each of the network's weights receives an update proportional to the partial derivative of the error function with respect to the current weight in each iteration of the training. The problem is that in some cases, the gradient will be so small that it will prevent the weight from changing its value. In the worst case, this may completely stop the neural network from further training. To overcome this problem, LSTM have been created.

4.3 Architecture of a Long Short-Term Memory

LSTM is a type of recurrent neural network which is able to learn and remember over long sequences of input thanks to some gates that control the flow of information through the network. LSTMs are able to solve the following RNNs problems [30] :

- **Short-term memory** : RNNs remove some information from earlier temporal phases when moving to later phases, resulting in the loss of important information. In contrast, LSTMs are able to retain all previous sequences without having to delete any.
- **Vanishing gradient** : As explained previously, this problem appears when the gradient becomes so small that it does not apply any changes to the weights during the back-propagation.
- **Exploding gradient** : This problem occurs when unreasonable importance is given to the weights of the network during the learning process.

The structure of this network is similar to the one represented on Figure 4.10: it contains a hidden state h_t but also a cell state c_t that will be used to compute the expected output, $\hat{y}_t$. Moreover, several computational gates are introduced in order to control the flow of information. Those three gates are:

- the input gate i_t is a switch to transmit or not x_t
- the output gate o_t controls whether h_t depends or not on c_t
- the forget gate f_t controls whether c_t depends or not on c_{t-1}

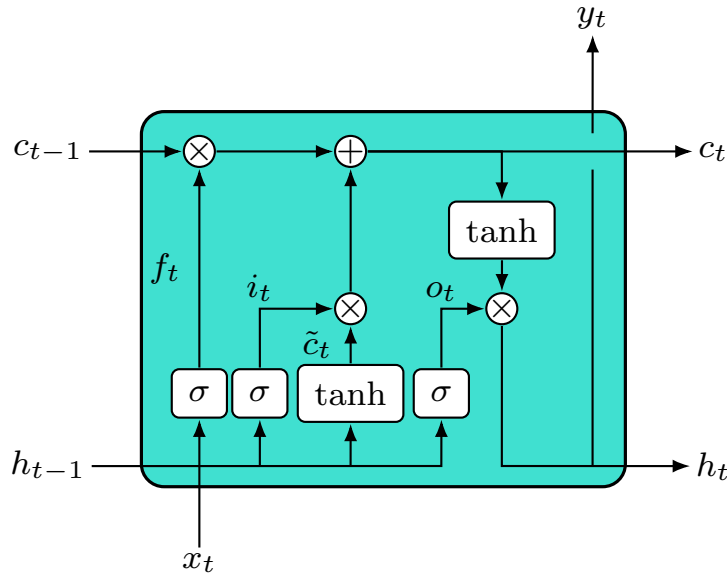


Figure 4.12: Representation of the architecture of a LSTM

These gates work as switches that may or may not let the signal go through the state cell. These are controlled by weights that are learned thanks to a back-propagation through time method. Therefore, LSTMs help to prevent both overfitting, where the model performs well on training data but poorly on test data, and problems related to the vanishing gradient. LSTMs can be described mathematically thanks to the following set of equations:

$$\begin{aligned}
i_t &= \sigma(W_{xi} x_t + W_{hi} h_{t-1} + b_i) \\
o_t &= \sigma(W_{xo} x_t + W_{ho} h_{t-1} + b_o) \\
f_t &= \sigma(W_{xf} x_t + W_{hf} h_{t-1} + b_f) \\
\tilde{c}_t &= \tanh(W_{xc} x_t + W_{hc} h_{t-1} + b_c) \\
c_t &= f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t \\
h_t &= o_t \cdot \tanh(c_t)
\end{aligned} \tag{4.22}$$

Where W_{mn} is the weight that controls the link between state m and gate n and b_n is the bias of gate n . Moreover, σ represents a non-linear sigmoid function that is applied to the input received by each gate. This will allow us to get a value between 0 and 1 that

tells if the gate should be open or not. Finally, $\tanh$ represents the application of a non-linear hyperbolic tangent activation function on the input. As we have seen previously, this function squishes values between -1 and 1.

Now that we have explained all the elements that compose the LSTM, let us see how information flows through an LSTM cell in practice [30]. The first step is to pass the previous hidden state h_{t-1} and information about the current input x_t through the forget gate f_t . Note that before being passed through this gate, data are squished between 0 and 1 because of the application of the sigmoid activation function. Therefore, if the resulting value is close to 1, the gate lets information flow and if it is close to 0, it indicates that information should be forgotten.

After that, the previous hidden state h_{t-1} and the information about the current input x_t are passed through the input gate i_t . Like for the previous step, before being passed through this gate, the data are given as input to a sigmoid activation function. In parallel, data are also given as input to a $\tanh$ activation function and the resulting value is represented by $\tilde{c}_t$. The two sets of resulting values that were computed in parallel are then multiplied together ($i_t \times \tilde{c}_t$) to get a single output.

The third step is to compute the next cell state. In order to compute this cell state, we need to multiply the previous cell state c_{t-1} with f_t , the result of the first step. After that, the result of this multiplication is then added to the result that we obtained at the end of the second step, resulting in a new cell state c_t .

In the fourth and last step, the previous hidden state h_{t-1} and the information about the current input x_t are passed through the output gate o_t . Before being passed through this gate, data are given as input to a non-linear sigmoid activation function. The resulting value is then multiplied with the value obtained by passing the new cell state c_t into a $\tanh$ activation function. This final resulting value represents the new hidden state h_t .

Finally, the new cell state c_t and the new hidden state h_t are then carried forward to the next time step.

4.3.1 Dropout and recurrent dropout

As with other machine learning models, RNNs can overfit. To address this problem, in addition to the use of LSTM, one can introduce dropout and recurrent dropout [31]. This allows to forget certain connection in the network and reduce overfitting, improving the performances. Using the keras implementation of LSTM, two types of dropout are available: `dropout` and `recurrent_dropout`.

On Figure 4.13, we observe that the x_t connection can be dropped to 0 when the `dropout` parameter is used. In the same way, we observe on Figure 4.14 that the hidden state of the previous time h_{t-1} can be dropped as well when the `recurrent_dropout`

parameter is used.

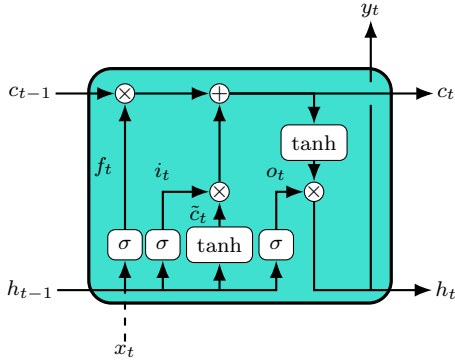


Figure 4.13: Connection dropped when using dropout

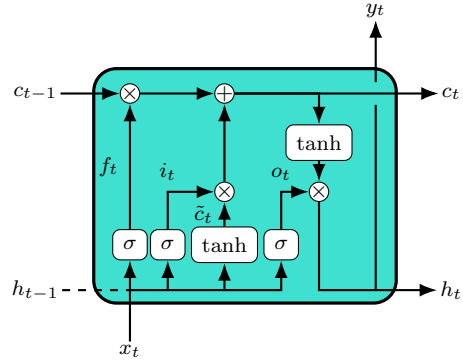


Figure 4.14: Connection dropped when using `recurrent_dropout`

4.4 Architecture of a Bidirectional Long Short-Term Memory

Bidirectional LSTMs are extensions of the classical LSTMs. Indeed, bidirectional LSTMs train two different LSTMs on the input sequence instead of one. The first one is trained on the initial input sequence and the second one is trained on a reversed copy of the input sequence. This method can provide additional context to the network and, therefore, lead to a better learning.

The difference between a bidirectional LSTM model and a unidirectional LSTM model is that the unidirectional LSTM is only able to preserve information from the past, i.e. information from the inputs it has already passed through using the hidden states [32]. On the contrary, the bidirectional LSTM model combines two hidden states: one from the LSTM that runs forward and one from the LSTM that runs backward. Since the first one provides information about the past and the second one about the future, a bidirectional is capable of preserving information from both past and future at any point of time. Therefore, a bidirectional LSTM model is able to understand the context of a situation in depth, unlike a unidirectional LSTM model.

Let us look at a concrete example to better understand the difference between these two models. We will consider the problem of embedding, i.e. mapping words into a numerical space, such that a computer can understand it. In this space, each word is encoded by a numerical vector. To be able to transform correctly the words into their numerical vector, a neural network needs to understand their context, as the same word can have multiple meanings. Now consider the two following sentences as a starting point:

- "I went to the river **bank**" [33]
- "I need to go to the **bank** to make a deposit" [33]

In the first sentence, the unidirectional LSTM model will be able to understand the context of the word *bank* because only the left context, i.e. the past context, is needed to deduce the meaning of this word. On the contrary, in the second sentence, the unidirectional LSTM model will not be able to understand the context of the word *bank* since the left context does not say anything about the meaning of this word. Indeed, we do not know if this word represents the place where one can sit by a river or if it represents rather the place where one can deposit money. The bidirectional LSTM model is able to understand the context of the second sentence since it has information from the left and the right context for any word. In the second sentence, the right context is the key to understand the meaning of the word *bank*.

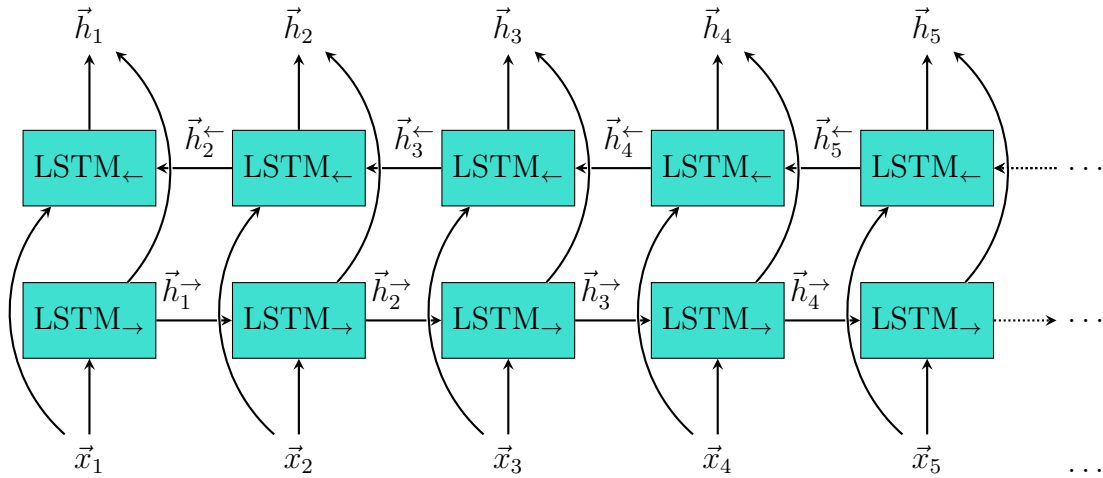


Figure 4.15: Representation of a bidirectional LSTM

4.5 Architecture of an Encoder-Decoder

"Autoencoders are an unsupervised learning technique in which we leverage neural networks for the task of representation learning. Specifically, we'll design a neural network architecture such that we impose a bottleneck in the network which forces a compressed knowledge representation of the original input.", J  r  my Jordan [34].

Those types of feed-forward neural networks have typically an hourglass shape, as shown in Figure 4.16. In short, this method is able to compress a multidimensional sequence into a single vector representing that information and to learn this compressed representation of the input. An encoder decoder (also known as autoencoder) is composed of three main elements :

- **Encoder:** this part of the network is responsible for compressing the input data.
- **Inner representation:** this is the bottleneck of the network. It contains the compressed representation of the input data.
- **Decoder:** this part of the network is responsible for decompressing the data contains in the inner representation. The goal of the decoder is to reconstruct the

original input data as best as possible.

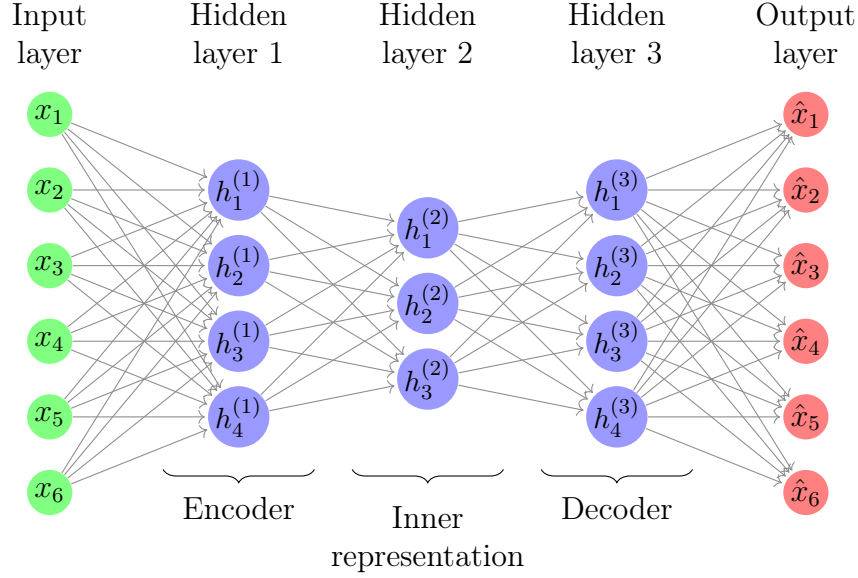


Figure 4.16: Representation of an encoder - decoder

For a time-series prediction task, it is advisable to use an LSTM Encoder-Decoder model. This is an implementation of an Encoder-Decoder model for sequence data using an Encoder-Decoder LSTM architecture. It means that the encoder part of the network can be composed of several LSTMs. In the same way, the decoder can be composed of several LSTMs as well.

A considerable advantage of models like these is that their output vector can have a different size in time compared to the one given as input. This is not the case for LSTM layers, only able to provide the same number of time steps or a single time step. For temporal prediction tasks, being able to output a different size can be very useful. Indeed, let us take the example of a weather prediction. If the goal is to predict the weather for the 5 next days, the Encoder-Decoder model is able to take the information about the weather of the 15 previous days. This is not the case for most models since in general, models such as simple LSTMs use the information about the last 5 days to predict the weather for the next 5 days. In this case, the size of the input vector of the model is the same as the output vector.

The previous sections of this report have provided an understanding of how data is collected and prepared so that it can be used by a time-series prediction model. In chapter 4, we have theoretically discovered the neural networks that are currently available for this kind of prediction. In this chapter, we will describe the different prediction models that we used during our work to predict the number of new hospitalizations in Belgium due to COVID-19. The choice of the prediction model that is used is crucial to achieve satisfactory predictions. To do this prediction, we have adopted two different approaches which we will detail in this section.

Several variables will be used in order to characterize the prediction model. Let us describe these different variables in this introduction before getting to the heart of the matter :

`n_samples` represents the number of past days that are given as input to a model in order to deduce a prediction, i.e. the size of the time dimension of the input vector that is given to the prediction model.

`n_forecast` represents the number of next days that we want to predict, i.e. the size of the time dimension of the output vector produced by the prediction model

`window` is the number of days relative to each prediction, and is equal to `n_samples + n_forecast`. The input window depicts the first `n_sample` data points of the window, given as input and the output window denotes the `n_forecast` last points, given as output by a model.

5.1 Reference models

In order to be able to evaluate the performance of the prediction models we have implemented, we need comparison models that we can consider as benchmarks. These reference models will allow us to ensure that the results of our models are satisfactory. We decided to compare the results of our predictions with those of three different basic models. They will be briefly described in this section, but we will not dwell on them.

The graphs used to represent the behaviors of those reference models were constructed in a general way to give you the intuition behind each prediction model. The goal is to try predicting the next `n_forecast` values of a transformation of a cosine function using the last `n_samples` values of the signal. In order to keep the graphs readable, the values chosen were `n_forecast` = 10 and `n_samples` = 20. The transformation of the signal was

simply changing the range from $[-1, 1]$ to $[0, 1]$. In the next chapter, we will adapt these specific models to our problem, i.e. the prediction of the number of new hospitalizations in Belgium.

5.1.1 Linear regression model

This model uses a linear regression based on the target value of the last n days, with $n \in [2, n_samples]$, in order to predict the target value for the $n_forecast$ next days. This model is rather simple and must not go through a fit operation in order to be trained : each input window given to the model includes the relevant information. A representation of behavior of the model on a tiny example is shown on Figure 5.1. Several predictions on the first horizon are illustrated on Figure 5.2.

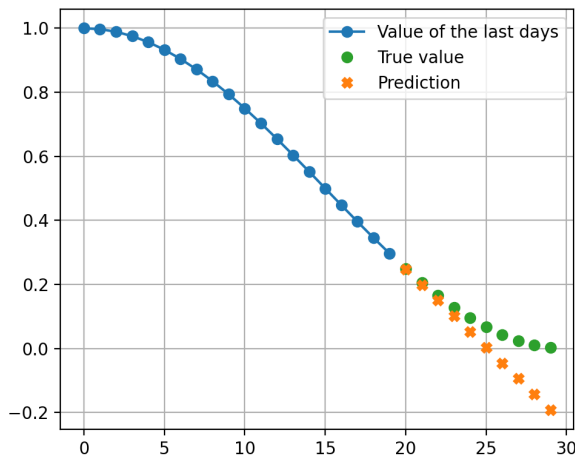


Figure 5.1: Expected prediction from a linear regression model on a window

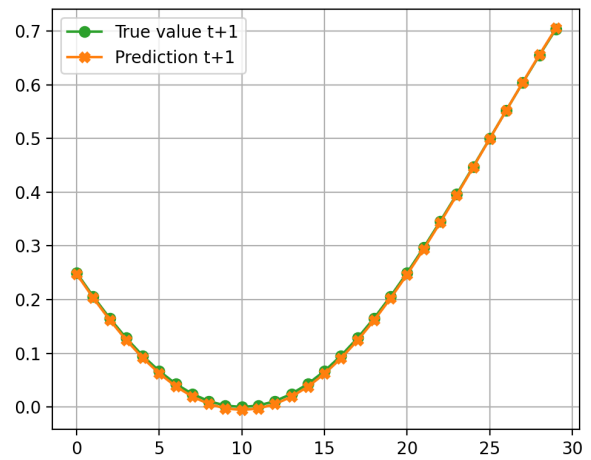


Figure 5.2: Expected prediction from a linear regression model for $t+1$

In this example, the very first point predicted is relatively close to the real value. However, when the horizon is high, the prediction worsen a lot. This can be explained by the fact that a linear regression can only make a linear type of prediction. However, the cosine function is of course nonlinear.

5.1.2 Repeat Last model

This model is very basic since it repeats the last value of the number of hospitalizations $n_forecast$ times. A representation of the behavior of the model is shown on Figure 5.3. Several predictions on the first horizon are illustrated on Figure 5.4.

In most machine learning techniques, a first reference model to beat is considered to be a linear regression. But when dealing with time series, this type of model tends to give better performances while still being simple and is therefore preferred [35, 36]. From Figure 5.4, we also see that this model consists simply of the lagged values of the target.

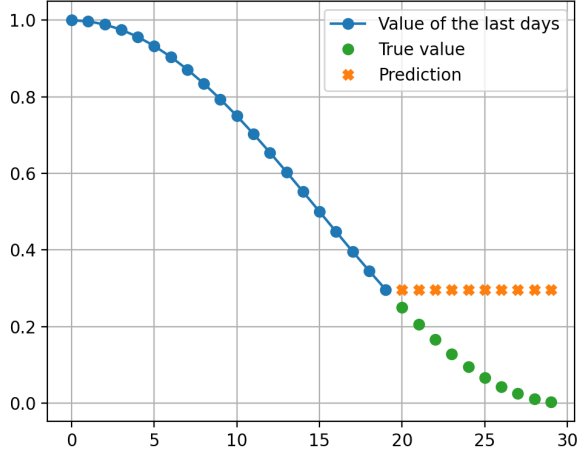


Figure 5.3: Expected prediction from a Repeat Last model on a window

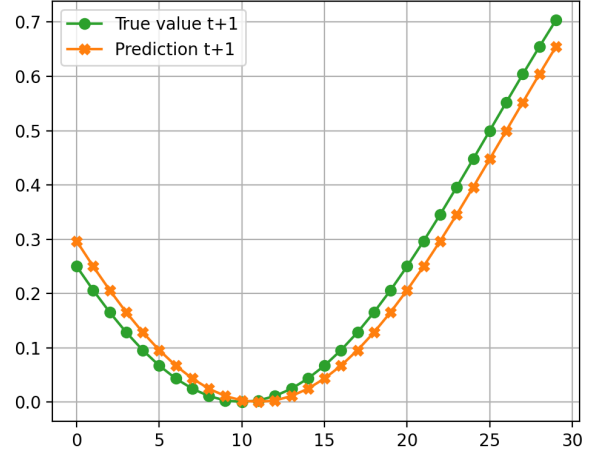


Figure 5.4: Expected prediction from a Repeat Last model for $t+1$

5.1.3 Dense model

Our last reference model to beat is only composed of one fully connected layer, i.e. a Dense layer. It selects the target values of the `n_samples` previous days in order to predict the target values of the `n_forecast` next days. An example of the type of predictions that we can expect from this model is shown on Figure 5.5. Several predictions on the first horizon are illustrated on Figure 5.6

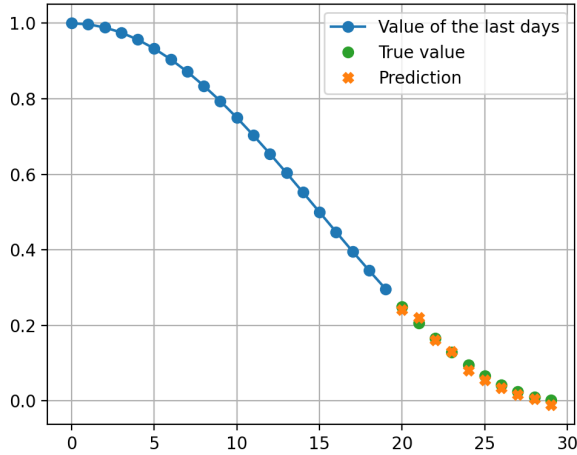


Figure 5.5: Expected prediction from a Dense model on a window, when trained using a similar signal as the one that needs to be predicted

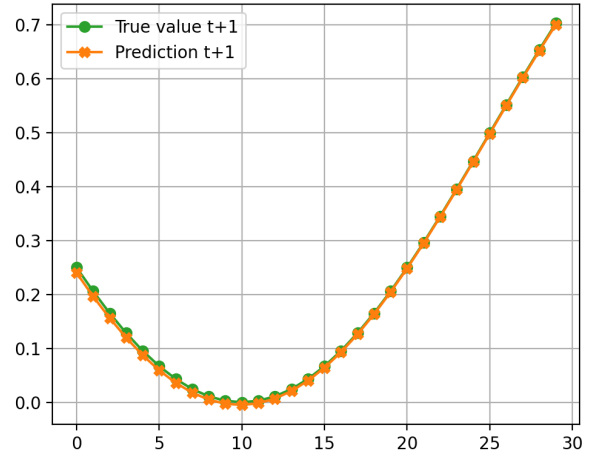


Figure 5.6: Expected prediction from a Dense model for $t+1$, when trained using a similar signal as the one that needs to be predicted

On this particular example, the model is able to predict relatively correctly what will happen in the next days. However, as for other models, the prediction tends to worsen on the last horizons. This example was also rather simple as it consists of a variation of a cosine signal. The signal that we are dealing with, i.e. the hospitalizations signal, is much harder to deal with. In contrast to the two previous model, the predictions points

do not consist of a simple line.

This model can behave differently depending on the parameters that it uses. The training of the model whose predictions are shown on Figures 5.5 and 5.6 was based on a hundred of epochs, consisting of data points similar to the one shown here. It is possible to gain more performance by increasing the number of epochs because the problem illustrated in the graph is relatively simple.

5.2 First approach

The first approach describes the first prediction model we built. The aim of this approach is to use an encoder-decoder model for the prediction. In contrast to the reference models described above, this model does not only take data related to the absolute number of past hospital admissions as input. Indeed, the input data of the model are the following :

- **Google Trends data** : All the *Google Trends* data from February 2, 2020 are given as input of the model.
- **Hospitalization data** : All the past total and new hospitalizations data are given as input of the model. Remind that the new hospitalizations depicts the variation of the total hospitalizations, and can therefore be negative or positive.
- **Transformed data** : Some more data are given as input of the model to facilitate its understanding of the links between the data. For example, the logarithmic transformation of the total number of hospitalizations is given as input (see section 3.2).

Note that all data that are given as input to the prediction model are considered from February 2, 2020. Moreover, each input window contains those types of data on the last `n_samples` days.

The purpose of this neural network will be to learn by itself how important the *Google Trends* data is. Conversely, if this data does not allow the model to obtain better predictions, it will decide by itself not to use it for the prediction. With this type of model, it is difficult to know exactly what impact the *Google Trends* data has. One way to analyze this will be described later on in section 6.2.

The architecture of the prediction model we built is relatively simple. This is a sequential neural network composed of an encoder, an inner representation and a decoder part, as explained theoretically in the section 4.5. The encoding part is itself composed of two different layers of *LSTM* which allows the network to encode the input data. These two layers are followed by a *RepeatVector* layer that repeats the encoded representation a specific number of time, typically `n_forecast` times. The third part of the model is the decoding one. The decoder is composed of two different layers of *LSTM*. The last layer of

the model is a *TimeDistributed(Dense)* one that allows us to apply a *Dense* layer to every temporal slice of the input data. In other words, the *TimeDistributed* layer will apply the same *Dense* layer to every time step during the unrolling of the *LSTM* cell. The dense layer is composed of 1 unit, in order to output 1 feature : the new number (or variation) of hospitalizations.

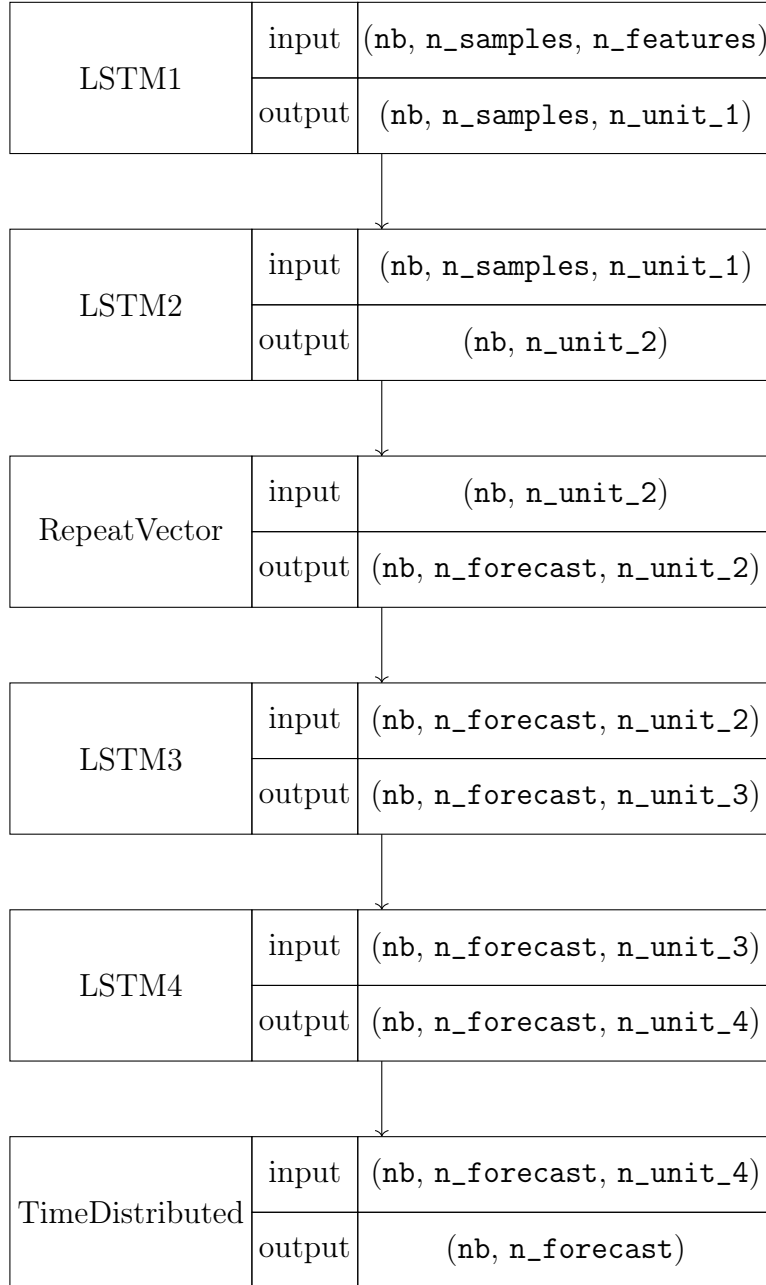


Figure 5.7: Representation of the architecture of the model. **n_features** is the number of features given to the model (both hospitalizations and *Trends* features) and **n_unit_i** is the number of units composing the layer with number **i**. **nb** denotes the number of samples given to the model.

The number of units for the different layers of Figure 5.7 are set to :

`n_unit_1 = 16`

`n_unit_3 = 4`

`n_unit_2 = 4`

`n_unit_4 = 16`

Note that these parameters were not fixed randomly, they have been defined by the tuning of hyper-parameters that will be described in the next section.

5.2.1 Tuning of hyper-parameters

In machine learning, a hyper-parameter is a parameter whose value is used to control the learning process. In contrast, the values of the other parameters are obtained by learning. However, the time required to train and test a machine learning algorithm can depend on the choice of its hyper-parameters [37]. Therefore, these hyper-parameters must be chosen carefully to avoid the model becoming too time-consuming.

Deep learning models are full of hyper-parameters that need to be fixed but finding the best hyper-parameters can be difficult in a high-dimensional space. In order to find it, we decided to use *Talos* [38] which is a hyper-parameter optimizer that can be used with *Tensorflow* [39]. Here are the hyper-parameters that we consider in order to find the best ones :

- **activation** : The activation function that is used in order to activate neurons in the neural network. *Tested* : *relu*, *elu*
- **reg** : The regularization parameter of the kernel used to penalize model in high dimensional space. This changes the value of the cost, becoming the sum of the loss function and the value of the penalty. *Tested* : *l1*, *l2*
- **regw** : The weights of the regularization parameter of the kernel. *Tested* : $1e^{-4}$, $5e^{-4}$, $1e^{-3}$
- **optimizer** : The optimizer is used to solve optimization problems by minimizing the objective function. It is used to update the attributes of the neural network such as weights and the learning rate in order to minimize the loss. *Tested* : *Adam*, *RMSprop*
- **losses** : Since we try to minimize an objective function, we need to define how to compute this function. *Tested* : *mse*, *mae*
- **epochs** : The number of epochs that is used to fit the model. *Tested* : *3000*, *5000*, *7000*, *10 000*
- **dropout** : The rate of units that needs to be ignored during the training phase of a particular set of neurons which is chosen at random. This is a regularization method used to prevent overfitting. *Tested* : *0*, *0.1*, *0.2*

- **recurrent_dropout** : The rate of connections between the recurrent units that needs to be ignored. This is a regularization method used to prevent overfitting.
Tested : 0, 0.1, 0.2

5.3 Second approach

The second approach is a little bit different than the previous one. The goal here is to control the importance that is given to *Google Trends* data in order to improve the initial prediction, i.e. the prediction based solely on hospitalizations data. Therefore, we have first computed a prediction $\hat{y}_{t+i}^H \quad \forall i \in 1..n_forecast$ based only on the number of new hospitalizations of the past **n_samples** days. This prediction is a basic estimate that we will try to refine slightly using data collected through *Google Trends*. This first estimation is constructed by a model that must not be trained, such as a linear regression or a baseline model.

From this simple prediction, we compute a correction factor C_{t+i} which is the ratio between the real value y_{t+i} that needs to be predicted and the value $\hat{y}_{t+i}^H$ given by the first model, based on the hospitalizations. This can only be done on the training data set, as we need the true value y_{t+i} to compute it.

$$c_{t+i} = \frac{y_{t+i}}{\hat{y}_{t+i}^H} \quad (5.1)$$

We then train a machine learning model, using only the data relative to the *Trends*, and predicting the correction factor. This prediction $\hat{c}_{t+i}$ is then used together with the prediction from the first model to construct the final result

$$\hat{y}_{t+i} = \hat{c}_{t+i} \times \hat{y}_{t+i}^H \quad (5.2)$$

However, there are some practical considerations with this technique. The correction factor can sometime be really large, for instance on the first days of the crisis, where the relative change in hospitalizations was huge, or on high horizons, where the first predicted values can be far away from the prediction. In the perfect scenario, the correction factor should be close to 1. We have therefore applied a threshold function, in order to limit the correction factor that needs to be predicted. Moreover, to have an even easier target to predict, the model does not predict directly the correction factor but rather $c_{t+i} - 1$, to have an output close to 0. The values that the model is trying to predict are therefore defined as :

$$\alpha_{t+i} = f(c_{t+i}, threshold) - 1 \quad (5.3)$$

with

$$f(c_{t+i}) = \begin{cases} 1 + threshold & \text{if } c_{t+i} > 1 + threshold \\ 1 - threshold & \text{if } c_{t+i} < 1 - threshold \\ c_{t+i} & \text{otherwise} \end{cases} \quad (5.4)$$

The model thus provides a prediction $\hat{\alpha}_{t+i}$, which is used to enhance the initial prediction and compute $\hat{y}_{t+i} = (\hat{\alpha}_{t+i} + 1) \times \hat{y}_{t+i}^H$. We have decided to set the *threshold* to 0.3, as the *Trends* data should not be allowed to have a huge impact on the prediction but rather change slightly the results. An example of the behavior of the model is shown on Figure 5.8.

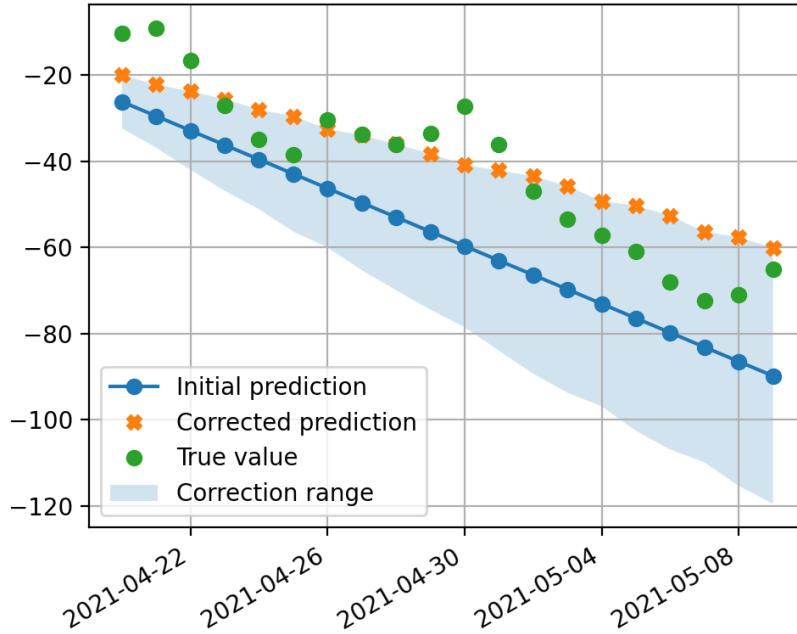


Figure 5.8: Behavior of the second approach, where an initial prediction is corrected using the *Trends* data. The initial prediction (blue) is allowed to change its value using a correction factor. This correction factor cannot exceed a certain range (bluish area), set by a threshold

In Figure 5.8, we see that the initial prediction is in blue. This prediction is made by a basic model using only the hospitalization data. Then, based on the *Google Trends* data, we will be able to correct this prediction in the range that is shown in light blue on the figure. Thanks to this correction, the final prediction made is shown in orange. We observe that the final prediction is closer to the true value than the initial prediction for most points.

So far we have explained how to collect the data and how to transform it so that it can be used by a prediction model. Then we have described the prediction models that we will use in order to predict the number of new hospitalizations. The next step is to generate predictions from those data and models and evaluate how accurate they are. To do this, we need to introduce an evaluation metric. In this section, we will therefore analyze the performance of each prediction model against the chosen evaluation metric. We will also try to find which features relative to *Google Trends* data are the most important ones, i.e. should be taken into account as input of the prediction model in order to produce the best possible results.

6.1 Evaluation metric

The most basic approach to evaluate the performance of a prediction model is to use the classical way of evaluation that is represented on Figure 6.1. In this approach, we need a training, a validation and a test set.

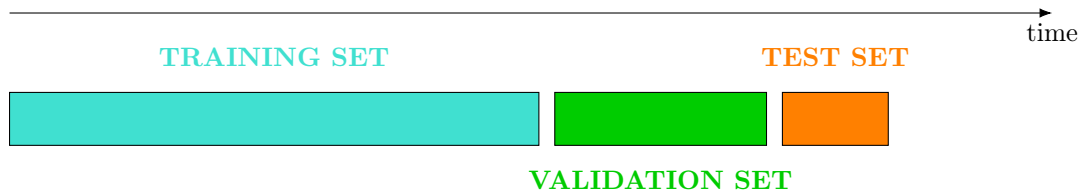


Figure 6.1: Representation of a classical way of evaluation

In time series forecasting, predictions become less and less accurate over time. Therefore, it is more realistic to re-train the model with actual data as it becomes available for new predictions. In this way, we are able to evaluate the performance of our model as if we were in the case of a real-time prediction. That is why we chose to evaluate the performance of our prediction model with a *walk forward* evaluation method [40, 41]. This also mimics what would happen in a real case: a model is retrained several time with the newest data, in order to account for changes in the dynamics of the system. Figure 6.2 shows how a walk forward evaluation looks like.

On Figure 6.2, we represented three different walks. Each of them is composed of a training set, a validation set and a test set. In practice, the model is trained on the training set and produces a prediction on the test set. This prediction is then evaluated to determine the quality of the prediction model. We also used a validation set during the simulations in order to be able to apply early stopping. This means that when a chosen

metric does not improve during p (patience) epochs, the simulation is stopped because we consider that the optimum is reached. If we continue to iterate in this kind of situation, we risk overfitting.

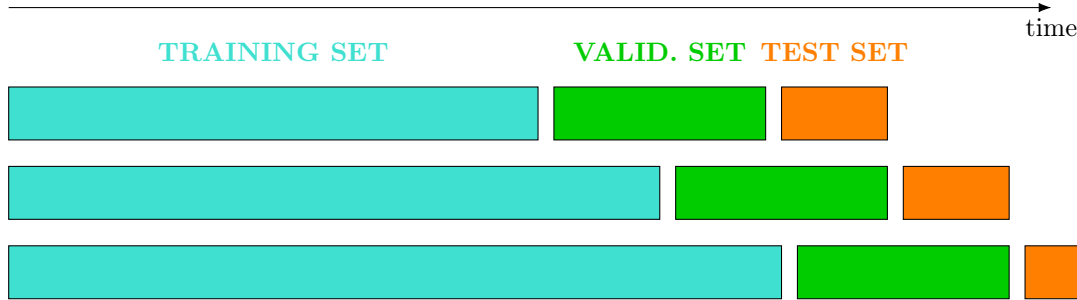


Figure 6.2: Representation of a walk forward evaluation

Concerning the computation of the error, we decided to evaluate the MSE and the MAE on the test set for each walk and for each horizon. An example of the dataframe corresponding to the final error is provided in Table 6.1. In this example, we assumed that a two-day prediction was made. Indeed, we observe the values of MSE and MAE for horizons $t + 1$ and $t + 2$. We estimated the error on three different walks to be consistent with the scheme shown in Figure 6.2.

	MSE (t+1)	MAE (t+1)	MSE (t+2)	MAE (t+2)
Walk				
walk 0	596.99	13.06	1226.66	18.51
walk 1	992.04	15.21	2196.83	23.34
walk 2	399.01	10.14	681.80	14.61

Table 6.1: Example of a dataframe of error on each walk

To simplify the interpretation of the results, we will match each walk with a time period corresponding to one month. However, the predictions are done on windows and one window can include dates covering several months. Because of this, we have to decide which month each window depends on. Unfortunately, some windows contain dates that cover several different months. Therefore, we decided to build all windows and separate them into several groups, or periods, when a window is able to cover a whole month in its forecasting dates. More specifically, with `n_forecast = 30` days of predictions and using `n_samples = 30` days to construct the predictions, the windows on the periods related to April 2021 and May 2021 are:

April 2021 : this period includes the following windows:

- First window considered: forecast on 2021-04-01 up to 2021-04-30, using the data from 2021-03-02 to 2021-03-31
- ...

- Last window considered: forecast on 2021-04-30 up to 2021-05-29, using the data from 2021-03-31 to 2021-04-29

May 2021 : this period includes the following windows:

- First window considered: forecast on 2021-05-01 up to 2021-05-30, using the data from 2021-04-01 to 2021-04-30
- ...
- Last window considered: forecast on 2021-05-31 up to 2021-06-29, using the data from 2021-05-01 to 2021-05-30

In this example, we see that the first window of April contains all the forecasting dates of that month. As for the last window, we observe that the forecasting dates considered go from 2021-04-30 up to 2021-05-29. These dates therefore cover the whole month of May except for the last day. The next window will cover the whole month of May and will therefore be independent of the April 2021 period. This window is the first window observed for the May 2021 period.

As data may not be available for the latest dates in the last month, we will use as many data points as possible to test these periods. Those periods can then be used in a walk forward evaluation. For instance, with one period of validation and one period of test, a possible walk could be:

Training set : includes the period before the validation set, i.e. related to

- First training window: forecast on 2020-03-02 up to 2020-03-31, using the data from 2020-02-01 to 2020-03-01
- ...
- Last training window: forecast on 2021-02-28 up to 2021-03-29, using the data from 2021-05-01 to 2021-05-30

Validation set : includes the period before the test set, i.e. related to March 2021, with the following evaluation windows:

- First validation window: forecast on 2021-03-01 up to 2021-03-30, using the data from 2021-01-30 to 2021-02-28
- ...
- Last validation window: forecast on 2021-03-31 up to 2021-04-29, using the data from 2021-03-01 to 2021-03-30

Test set : includes the period related to April 2021, with the windows detailed just before.

Another particular walk can also be created before or after these periods in order to evaluate the performance of the model on a month other than April 2021.

6.2 Feature importance

We have a total of 12 different topics that we can collect and use in each region. The purpose of this section is to define which topics are the most important, i.e. are significantly useful at predicting a target variable. This will provide some insights on what *Google* searches are the most relevant for the prediction of hospitalizations. There are three different ways to do it that will be presented in this section.

For each method, we will assess the importance of the features based on a *walk forward* evaluation metric. In contrast to what was described in the previous section, this evaluation method does not use a validation set. We have also decided to modify the use of the training set. On Figure 6.3, you can observe the principle of this evaluation metric. The size of the training and test set is fixed and at each walk, both sets are shifted by the length of the test set. We have decided to train a model using four months of data, corresponding roughly to 120 training windows on each country/region. We will then assess the performance of this model on the next month, which includes about 30 test windows. This choice was done as the importance of features evolves over time and it is therefore important to be able to define which features were important at which time. For example, at the very beginning of the crisis, the topic "anosmia" was certainly a popular topic for people because those who lost their smell did not know what this symptom meant. Afterwards, everyone knew that the anosmia was a major symptom of coronavirus, so fewer and fewer people searched for it on a *Google* search engine [42].

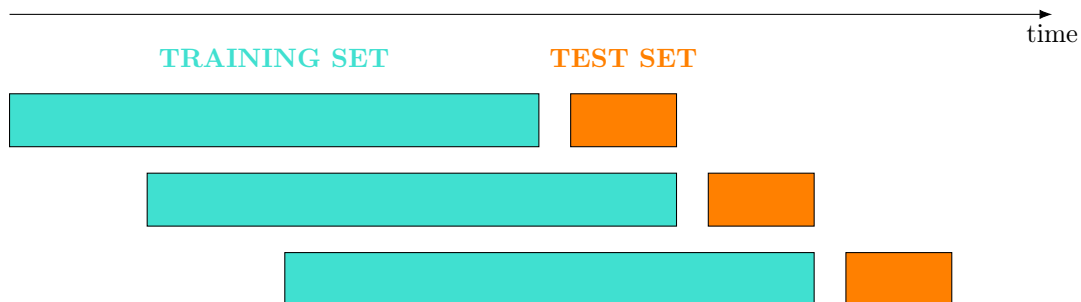


Figure 6.3: Representation of a walk forward evaluation for the feature importance

We now have all the information we need to move on to the section dealing with feature importance. We will analyze three different methods to determine which features are the most important according to the period considered. These methods are also sensitive to the type of model used. Some of them depend on a specific architecture, others do not. This means that the results are likely to differ from one feature analysis method to another, as the method and the model used will be different.

6.2.1 Based on sensitivity analysis

This first method of assessing which features are the most important among those given as input to the prediction model is based on LIME [43] and is designed in two main steps. The first step is to make a prediction taking into account all the features we have

at our disposal and compute its error. This means that the model will be trained with data related to hospitalizations as well as with the *Google Trends* data we have collected. The second step consists of masking the data for each feature one by one in the input of the model, i.e. replace the interest over time of that feature with zeros in the dataframe. This masked input is then used to construct a new prediction, and the error is again computed. This last step is repeated for all features. With this method, we will be able to find out the impact on the quality of the prediction when we delete each feature. Indeed, if the prediction is similar to the initial prediction after deleting the impact of a particular feature, we can deduce that it is of little importance and has a low impact on the target variable. On the other hand, if the prediction becomes poor when a particular feature is deleted, we can deduce that this feature is important for the prediction of the target variable.

To simplify the understanding of this method, let us illustrate the principle with a concrete example. For our example, we will consider that `n_samples` is equal to three. This means that the data of the last three days will be used by the model to make the prediction. Therefore, the initial dataframe that will be given as input to the prediction model will contain data with a lag of three days (see section 3.2.3 for more information). Moreover, we will assume that we have only two features and we want to determine which one is the most important. These two features are "Fever" and "Cough". The initial dataframe containing the interest over time of these features can be found in Table 6.2. This dataframe will be given as input to the prediction model as well as hospitalization data. Therefore, we will get the initial prediction of the model with these data. The first step is over.

		Fever(t-2)	Cough(t-2)	Fever(t-1)	Cough(t-1)	Fever	Cough
LOC	DATE						
BE	2021-04-16	18.70	17.16	18.55	17.16	17.63	17.02
	2021-04-17	18.55	17.97	17.63	17.97	17.13	15.99
	2021-04-18	17.63	17.02	17.13	17.02	16.92	15.11

Table 6.2: Example of an initial dataframe that could be given as input of the prediction model

We can now move on to the second step. As explained theoretically earlier, we need to remove the role of each feature one by one. Let us start by removing the role of the feature "Fever". In Table 6.3, you can see what the new dataframe looks like. We will then give this new masked data as input to the model and get a new prediction. If the new prediction is bad, i.e. if the error of the model is higher than the first prediction, then we can deduce that "Fever" is an important feature. The same process will be applied to the feature "Cough" in order to check if this feature is important for getting a good prediction or not.

LOC	DATE	Fever(t-2)	Cough(t-2)	Fever(t-1)	Cough(t-1)	Fever	Cough
BE	2021-04-16	0	17.16	0	17.16	0	17.02
	2021-04-17	0	17.97	0	17.97	0	15.99
	2021-04-18	0	17.02	0	17.02	0	15.11

Table 6.3: Example of the dataframe when removing the role of the feature "Fever"

We have chosen to apply this method with an encoder-decoder model, as it uses LSTMs. For the time being, neural networks composed of LSTMs seem to be the most efficient architecture for making time series predictions [44]. Based on the walk forward evaluation metric that was defined previously, we are able to determine the most important features for each walk, i.e. for each month since August 2020. In order not to overload the results, we will present only the five most important topics for each walk.

	top 1	top 2	top 3	top 4	top 5
Aug 2020	PCR	COVID 19 testing	Symptom	Cure	Coronavirus disease 2019
Sep 2020	COVID 19 testing	Coronavirus disease 2019	Vaccination	Cure	Symptom
Oct 2020	COVID 19 testing	Sore throat	Cough	Coronavirus disease 2019	Vaccination
Nov 2020	Symptom	Coronavirus disease 2019	Virus	Fever	PCR
Dec 2020	Vaccination	COVID 19 vaccine	Virus	Allergy	PCR
Jan 2021	COVID 19 vaccine	Virus	PCR	Symptom	Coronavirus disease 2019
Feb 2021	COVID 19 vaccine	Allergy	PCR	Vaccination	Cure
Mar 2021	COVID 19 testing	Allergy	Coronavirus disease 2019	Symptom	Virus
Apr 2021	Coronavirus disease 2019	COVID 19 vaccine	Symptom	Fever	Sore throat
May 2021	Vaccination	Coronavirus disease 2019	COVID 19 vaccine	Allergy	Cough

Table 6.4: 5 most important features based on sensitivity analysis on each walk - New number of hospitalizations

In Table 6.4, we observe that there are many features that appear at least once in the top five of the most interesting topics during a particular walk. It is therefore difficult to deduce which topics are the most important ones in general. Nevertheless, we can first note that the topic "Coronavirus disease 2019" seems to be one of the most important ones since it appears one time as the prime importance topic and seven times elsewhere in the top five.

We also observe that terms related to testing are very present in the table. Indeed, we see that the topic "PCR" appears regularly in the top five of the most important topics. The topic "COVID 19 testing" appears even more often than the topic "PCR" in Table 6.4. We note that these terms are important at every moment. Nevertheless, it is clear that the importance of these terms was very high between August 2020 and October 2020. This can be explained by the fact that the number of tests performed increased exponentially from mid-August to October 2020 in Belgium [45]. The strong increase in the number of tests started around August 15, this may explain the fact that the topic "COVID 19 testing" is of second importance and not of first importance for the whole month of August 2020.

Moreover, features related to the vaccination are also very present. Indeed, we observed that the topics "Vaccination" and "COVID 19 vaccine" regularly appear in the top five most important topics. We notice that these topics become the most important ones from the month of December 2020. This can be explained by the fact that the vaccination campaign started in January 2021 in Belgium. Therefore, the media was talking a lot about the vaccine the month before, leading people to search for information about the coronavirus vaccination on the *Google* search engine. In addition to that, the vaccination campaign started at the end of December in France [46]. The results presented in Table 6.4 were found thanks to a mixture of data from Belgium and some French regions, so it is normal to observe a reflection of the situation in both countries.

Finally, we observe several symptoms that appear regularly in Table 6.4. These symptoms are typical of the coronavirus. We observe that the terms "Fever", "Sore throat" and "Cough" appear most often in the top five of the most important topics. We can therefore hypothesize that people suffering from these symptoms are searching on the *Google* search engine to check if they have a chance of having (unfortunately) caught the coronavirus.

6.2.2 Based on mean decrease in impurity

In order to find the most important features among those of the input data, we will use a *RandomForestRegressor* model [47]. This algorithm provides feature importance scores based on the mean decrease in impurity within each of its decision tree [48]. In other words, it measures how each feature is used in each decision tree to correctly split the samples. The more it is used and the better it splits, the more important the feature is. Because a random forest uses several decision trees, the importance score is computed as the average feature importance score over all trees.

In a first step, the aim of this method is to determine the importance of the lagged features for each topic for all the initial regions, i.e. Belgium as well as the non-augmented French regions. Next, the goal is to determine in general which topics are most important and should be given as input to the prediction model. To do this, we will simply calculate the importance of a particular topic as the average of the importance of each lag variable related to that topic weighted by the population of the region that is considered.

Based on the walk forward evaluation metric that was defined previously, we are able to determine the most important features for each walk. In order not to overload the results, we will present only the five most important topics for each period.

In Table 6.5, we first observe that the topic "Vaccination" seemed to be the most important one in August 2020. The most likely hypothesis to explain this is that, at that time, it was announced in Belgium that it was crucial to be able to vaccinate as many people at risk as possible against seasonal flu in order to avoid overburdening the health care system, given the possible risk of the flu season coinciding with a peak in COVID-19

[49]. After that, we observe that features related to vaccination such as "Vaccination" and "COVID 19 vaccine" consistently appear in the top five of the most important topics from October 2020. This can be explained by two different things. On the one hand, just as in August, the media reminded people of the importance of getting the seasonal flu vaccine for those at risk. On the other hand, the vaccination campaign against COVID-19 started in January 2021 in Belgium. This may explain why the term "Vaccination" is the most important one for this month.

Period	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5
Aug 2020	Vaccination	Symptom	Virus	COVID 19 testing	Cure
Sep 2020	Allergy	Cough	Fever	Coronavirus disease 2019	Sore throat
Oct 2020	PCR	Symptom	Vaccination	Fever	Mal de gorge
Nov 2020	Sore throat	Cough	COVID 19 vaccine	Virus	PCR
Dec 2020	Coronavirus disease 2019	COVID 19 testing	Symptom	Fever	PCR
Jan 2021	Vaccination	Coronavirus disease 2019	COVID 19 vaccine	Cure	Sore throat
Feb 2021	Cure	Coronavirus disease 2019	Vaccination	Virus	COVID 19 vaccine
Mar 2021	COVID 19 vaccine	COVID 19 testing	Vaccination	Fever	Symptom
Apr 2021	PCR	Fever	COVID 19 vaccine	Cure	Virus
May 2021	Symptom	PCR	COVID 19 vaccine	Coronavirus disease 2019	Vaccination

Table 6.5: 5 most important features based on mean decrease in impurity on each walk - New number of hospitalizations

We also observe the importance of terms related to testing. Indeed, in October 2020, the most important term was "PCR". This can be explained by the fact that this is when the number of tests performed related to COVID-19 in Belgium increased exponentially [50, 45]. The topic "COVID 19 testing" is also widely present in Table 6.5.

Finally, we observe several symptoms of coronavirus that appear as the most important features. For example, in October 2020, the symptom of "Sore throat" was the most important term. We also note that other symptoms such as cough and fever appear several times in Table 6.5. The topic "Symptom" also appears regularly in the table, so we can hypothetically deduce that people continue to search for symptoms related to the coronavirus on the *Google* search engine when they get sick at any time, not only at the beginning of the health crisis.

6.2.3 Based on feature permutation

The principle of this method is very simple. The goal is to define the decrease in the prediction quality of the model when the values of a feature are randomly shuffled. This method allows us to break the relationship between the feature and the target [51]. Therefore, if the prediction quality becomes poor, i.e if the error increases, we can assume that the target variable has a high correlation with the feature that has been shuffled. Moreover, this method is independent of the prediction model that is used.

This method is similar to the first one described in section 6.2.1. The difference between the two methods is that the sensitivity analysis method determines the importance of a feature by setting all data related to that feature to zero whereas this method will

shuffle the values of interest over time of a topic to measure its importance.

Based on the walk forward evaluation metric that was defined previously, we are able to determine the most important features for each walk. In order not to overload the results, we will present only the five most important topics for each walk.

Period	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5
Aug 2020	Vaccination	Symptom	Virus	Cure	COVID 19 testing
Sep 2020	Allergy	Sore throat	Fever	Cough	Coronavirus disease 2019
Oct 2020	Symptom	PCR	Fever	Vaccination	Sore throat
Nov 2020	Sore throat	Cough	PCR	COVID 19 vaccine	Cure
Dec 2020	COVID 19 testing	Symptom	Coronavirus disease 2019	Sore throat	PCR
Jan 2021	Coronavirus disease 2019	Vaccination	Cure	COVID 19 vaccine	Sore throat
Feb 2021	Cure	COVID 19 vaccine	Coronavirus disease 2019	Allergy	Virus
Mar 2021	COVID 19 vaccine	Vaccination	Symptom	Fever	Allergy
Apr 2021	PCR	Fever	Symptom	Virus	COVID 19 vaccine
May 2021	Symptom	PCR	Vaccination	Coronavirus disease 2019	COVID 19 vaccine

Table 6.6: 5 most important features on each walk - New number of hospitalizations

In Table 6.6, we observe very similar results to those obtained in the previous section relative to feature importance based on mean decrease in impurity (section 6.2.2). We will not comment on them in detail as this has already been done previously.

6.3 Prediction of the new hospitalizations

In this section, we will try to predict the number of new hospitalizations, i.e. the variation of people who are at the hospital because of COVID-19 at a particular time. Recall from section 2 that the number of new hospitalizations can be computed as the difference between the number of people admitted to hospital and the number of people discharged from hospital. The error values are computed based on the non-augmented data set. Note that we will perform the simulations for the following particular parameters: `n_forecast` = 30 days of predictions and `n_samples` = 30 days used to deduce the prediction.

The prediction will also be done based on two different types of localization. The first type denotes the French regions, along with Belgium, and the second type denotes the European countries. Nevertheless, we will only show the results for the French regions and Belgium in this section. The results obtained on the basis of the European countries will be shown in the appendices of this document (appendix D).

Because our predictions are done for up to 30 days of horizon, the tables are quite large to show. We therefore only present the tables regarding the MAE, as those number are the easier to interpret: an MAE of x on a given horizon means that, on average, the prediction model was x hospitalizations away from the true number for this horizon. However, in case the reader wants to know more details, a table presenting the MSE will

be shown in the appendix C.

To make the results more visual, we will also present graphs of the prediction of Mars 2021 in Belgium. Although it is possible to provide prediction graphs for every region and every window, only the ones regarding Belgium will be shown, as our objective remains the prediction of the number of hospitalizations in Belgium. Concerning the window that was chosen, we decided to represent only the predictions for the month of March 2021, this choice was arbitrary. Some graphs will show the evolution of the number of new hospitalizations due to COVID-19 in Belgium while others will represent the number of total hospitalizations. Nevertheless, we have tried to predict new hospitalizations and not total ones. This choice has been made to be sure that the values that our models are trying to predict on the test and validation sets are close to the values present in the training set. By combining the predictions of new hospitalizations over time, we can construct the prediction for the total number of hospitalizations.

As described previously in section 6.1, the model's performance are determined on the basis of a walk-forward evaluation. We have considered the last five months for the evaluation of our models: January 2021 up to May 2021 inclusive.

6.3.1 Reference models

The results of the three models that will be presented in this section are considered as benchmark results. The goal will be to achieve better predictions than these three models with the encoder-decoder model and the assembled model. As a reminder, the latter are the more complex models that we have built in order to produce accurate predictions, essentially in the long term.

Linear regression model

The aim of this model is to derive a good short-term prediction. Indeed, this model will use the last few points to fit a linear regression line and predict the next few days based on it. Nevertheless, the model is not intended to be used to predict a large number of next days. Intuitively, we therefore expect to obtain a fairly accurate prediction at $t + 1$. However, the prediction beyond $t + 5$ is expected to become extremely poor.

The results in Table 6.7 show that, on average, the prediction becomes less and less accurate as the number of prediction days increases. Indeed, we notice that the prediction at $t + (m + n)$ days (with $n \geq 1$) is always significantly worse than the prediction at $t + m$ days. This seems logical and corresponds to what we expected. We noticed that, on average, this model makes an absolute error equivalent to a little bit more than 26 hospitalizations for a 30-day prediction, which is less than expected. Indeed, we would tend to think that the error committed by the prediction would increase more significantly than what is observed with the horizon.

Table 6.7: MAE on each walk when predicting new hospitalizations for the linear regression model, for up to 30 horizons. The mean over all walks is also reported. Boldface indicates the best performance on each row. The training dataset covered the French regions and Belgium, composed of 23 initial regions and 156 augmented regions

Horizon	Jan 2021	Feb 2021	Mar 2021	Apr 2021	May 2021	mean
t+1	6.39	7.32	5.76	6.46	4.88	6.16
t+2	7.21	8.34	6.60	7.32	6.03	7.10
t+3	7.96	9.33	7.35	8.15	6.40	7.84
t+4	8.66	10.24	8.01	9.05	6.38	8.47
t+5	9.28	11.09	8.54	9.93	6.71	9.11
t+6	9.87	11.88	9.06	10.78	7.01	9.72
t+7	10.42	12.62	9.52	11.56	7.16	10.26
t+8	10.95	13.28	9.90	12.28	6.86	10.65
t+9	11.52	13.93	10.25	13.04	6.57	11.06
t+10	12.17	14.58	10.56	13.83	6.74	11.58
t+11	12.78	15.18	10.99	14.51	7.67	12.22
t+12	13.45	15.81	11.56	15.12	8.76	12.94
t+13	14.12	16.35	12.20	15.87	9.21	13.55
t+14	14.74	16.79	12.93	16.65	9.73	14.17
t+15	15.34	17.18	13.76	17.43	10.72	14.89
t+16	15.89	17.65	14.47	18.23	12.00	15.65
t+17	16.37	18.20	15.17	19.02	12.46	16.24
t+18	16.84	18.80	15.88	19.89	12.88	16.85
t+19	17.32	19.44	16.45	20.82	13.52	17.51
t+20	17.71	20.18	17.10	21.73	14.67	18.28
t+21	17.98	20.98	17.76	22.67	17.07	19.29
t+22	18.28	21.88	18.38	23.65	18.45	20.13
t+23	18.69	22.77	19.06	24.70	18.52	20.75
t+24	19.08	23.69	19.77	25.65	18.61	21.36
t+25	19.45	24.58	20.61	26.58	19.10	22.06
t+26	19.80	25.45	21.44	27.48	20.23	22.88
t+27	20.19	26.27	22.24	28.40	20.55	23.53
t+28	20.51	27.06	23.09	29.27	20.10	24.01
t+29	20.92	27.84	23.96	30.08	21.83	24.93
t+30	21.34	28.62	24.82	30.83	25.60	26.24

We also observe in Table 6.7 that the absolute error is the smallest for almost all horizons when it comes to evaluating the performance of the model over the month of May 2021. We can therefore assume that the predictions for the month of May are easier to derive for the linear regression model.

For a 10-days prediction, we observe on graph 6.5 that the predictions are not close to the true values. The shape of the prediction is not at all similar to the expected shape, i.e. the shape of the true values. Looking at graph 6.4, we also observe that even the prediction at $t + 1$ does not seem to be accurate at all, contrary to what we thought. By checking the values in Table 6.7, we observe that the one-day prediction still makes an

absolute error of more than 6 hospitalizations, which is high.

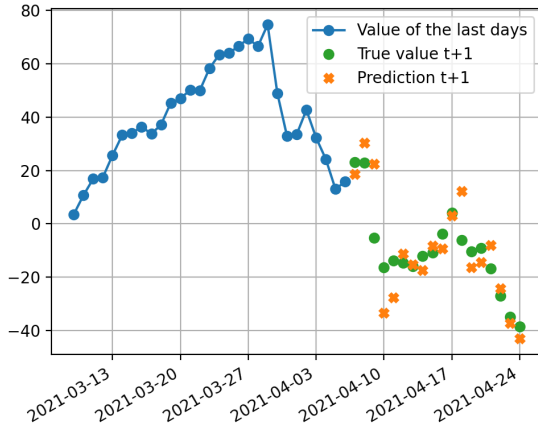


Figure 6.4: Prediction of the number of new hospitalizations in Belgium by a linear regression model compared to the true value at $t+1$

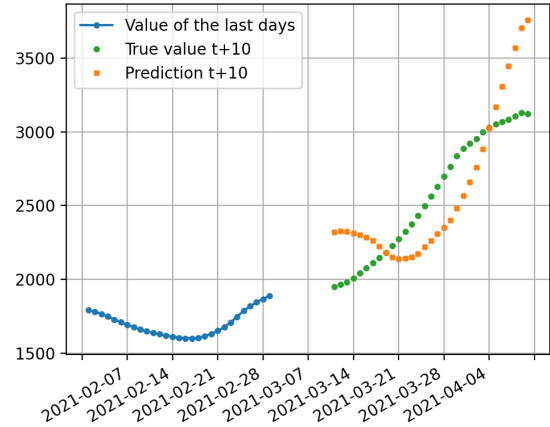


Figure 6.5: Prediction of the number of total hospitalizations in Belgium by a linear regression model compared to the true value at $t+10$

Baseline model

Since this model simply predicts the number of new hospitalizations as the same as the day before, we expect to get poor results. Indeed, the number of new hospitalizations can sometimes be a large number and therefore a constant prediction will have a larger error than if the number of new hospitalizations had been small. On the other hand, if there are few new hospitalizations, we expect to obtain a relatively small error.

At first glance, we could have thought that this model would have been the worst of all. After having done some simulations, we noticed that it was not the case. Indeed, this model achieves better results than the linear regression model on average and for each particular walk. The results in Table 6.8 show that this model makes an absolute error equivalent to a little bit less than 14 hospitalizations for a 30-day prediction. These results are significantly lower than those observed for the prediction with a linear regression model. With regard to the prediction of the next day ($t+1$), the baseline model had on average a smaller absolute error of nearly 4 hospitalizations than the linear regression model. Indeed, the baseline model makes an error of less than 3 hospitalizations for a one-day prediction. Concerning the 30-day prediction, the difference between the two models is much greater. In fact, the baseline model had a smaller absolute error of 13 hospitalizations than the linear regression model. On average, the baseline model makes an error of just over 13 hospitalizations for a 30-day prediction, which is really interesting.

Table 6.8: MAE on each walk when predicting new hospitalizations for the baseline model, for up to 30 horizons. The mean over all walks is also reported. Boldface indicates the best performance on each row. The training dataset covered the French regions and Belgium, composed of 23 initial regions and 156 augmented regions

Horizon	Jan 2021	Feb 2021	Mar 2021	Apr 2021	May 2021	mean
t+1	2.71	2.46	2.39	2.91	3.47	2.79
t+2	4.00	3.66	3.67	4.25	5.48	4.21
t+3	5.07	4.80	4.78	5.28	6.88	5.36
t+4	6.01	5.79	5.70	6.31	8.07	6.38
t+5	6.88	6.74	6.58	7.20	9.64	7.41
t+6	7.64	7.50	7.32	8.20	11.22	8.38
t+7	8.45	8.23	7.98	9.27	12.16	9.22
t+8	8.73	8.46	8.23	9.94	12.11	9.50
t+9	9.16	8.63	8.56	10.62	11.95	9.78
t+10	9.54	8.73	8.90	11.36	10.49	9.80
t+11	9.89	8.88	9.28	12.00	8.82	9.77
t+12	10.24	9.05	9.74	12.57	8.57	10.04
t+13	10.71	9.14	10.23	13.15	8.72	10.39
t+14	11.17	9.19	10.77	13.71	8.75	10.72
t+15	11.60	9.15	11.37	14.29	8.50	10.98
t+16	11.99	9.24	11.81	14.93	8.16	11.23
t+17	12.25	9.38	12.21	15.51	8.92	11.65
t+18	12.46	9.48	12.61	15.99	9.88	12.08
t+19	12.67	9.57	12.92	16.51	10.07	12.35
t+20	12.81	9.73	13.23	17.03	9.76	12.51
t+21	12.91	10.05	13.44	17.56	8.39	12.47
t+22	12.91	10.43	13.59	18.14	7.14	12.44
t+23	12.90	10.75	13.75	18.56	7.74	12.74
t+24	12.86	11.09	13.79	18.99	8.42	13.03
t+25	12.90	11.34	13.96	19.36	8.51	13.22
t+26	12.87	11.77	14.12	19.71	8.21	13.34
t+27	12.81	12.18	14.28	20.09	8.61	13.59
t+28	12.77	12.52	14.44	20.36	9.81	13.98
t+29	12.75	12.82	14.64	20.64	9.19	14.01
t+30	12.65	13.13	14.81	20.98	6.59	13.63

We did not expect such a good performance from this model. Nevertheless, after some considerations, we can explain why this model achieves such good results. This can be explained by the fact that the number of new hospitalizations is quite constant unlike the number of total hospitalizations. Therefore, estimate that the number of new hospitalizations for tomorrow will be the same as today is a relatively good approximation. On the other hand, if the number of new hospitalizations increases or decreases sharply from one day to the next, the model may become poor. Nevertheless, the number of new hospitalizations is often stable and rarely go from nothing to everything in only one day.

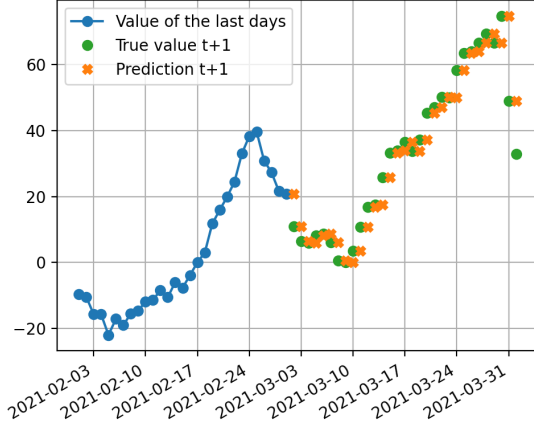


Figure 6.6: Prediction of the number of new hospitalizations in Belgium by a baseline model compared to the true value at $t+1$

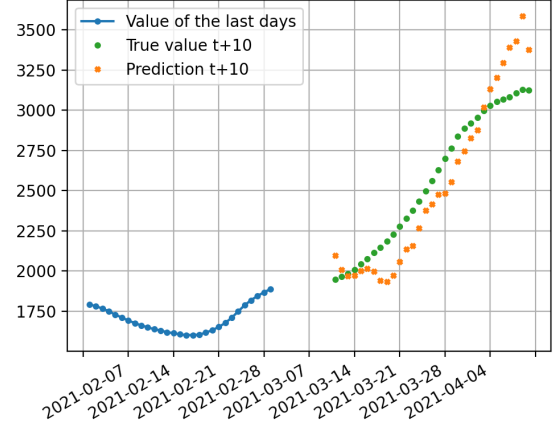


Figure 6.7: Prediction of the number of total hospitalizations in Belgium by a baseline model compared to the true value at $t+10$

On graph 6.6, we observe that the error committed by the model at time $t+1$ is very low, i.e. the predictions are very close to the real values. This can be explained by the fact that the graph shows the predictions that were made at horizon 1 in March 2021 in Belgium. Looking at Table 6.8, we see that among the 5 months on which we have tested the performance of our model, the month of March is the one that gives the smallest error at time $t+1$ on average. Therefore, we can conclude that the prediction for the month of March 2021 was simpler than for the other four months for this model.

Dense model

This model is a little more complex than the previous two since it requires training before it can make a prediction. Therefore, we expect this model to achieve better results than the first two models, especially for predictions far into the future. However, using a more complex model is not always a good idea when the prediction is simple. For short-term predictions, it is difficult to determine in advance which of the three reference models will be the most accurate.

Surprisingly, we notice that the error made for the prediction at $m+n$ days (with $n \geq 1$) is sometimes smaller than the error for the prediction at m days, which is counter-intuitive. For example, on the month of February 2021, from the 12-day prediction the error decreases with time until the 15-day prediction. This same situation sometimes occurs on other walks as well. We observe this same behavior in the average results. For example, the error committed for an 11-day prediction is slightly lower than the error committed for a 10-day prediction.

Table 6.9: MAE on each walk when predicting new hospitalizations for the dense model, for up to 30 horizons. The mean over all walks is also reported. Boldface indicates the best performance on each row. The training dataset covered the French regions and Belgium, composed of 23 initial regions and 156 augmented regions

Horizon	Jan 2021	Feb 2021	Mar 2021	Apr 2021	May 2021	mean
t+1	3.07	2.75	2.50	3.11	3.02	2.89
t+2	4.21	3.84	3.96	4.36	6.20	4.51
t+3	5.47	4.83	4.80	5.76	7.41	5.65
t+4	6.83	5.89	6.01	6.32	9.98	7.01
t+5	7.78	6.96	7.05	7.95	10.27	8.00
t+6	8.63	8.12	7.75	9.30	12.58	9.28
t+7	9.57	8.74	8.60	9.68	15.30	10.38
t+8	10.00	9.00	8.94	10.49	16.16	10.92
t+9	10.60	9.29	9.49	11.37	17.00	11.55
t+10	10.99	9.35	9.68	13.19	15.17	11.68
t+11	11.09	9.19	10.02	13.13	14.64	11.61
t+12	11.56	9.47	10.67	14.76	14.24	12.14
t+13	12.46	9.41	11.04	14.37	16.82	12.82
t+14	12.94	9.34	11.19	16.14	18.40	13.60
t+15	12.96	8.96	11.74	16.72	16.32	13.34
t+16	13.28	9.07	11.99	16.28	16.15	13.35
t+17	14.11	9.20	12.10	16.84	20.35	14.52
t+18	14.24	9.44	12.10	18.79	19.90	14.89
t+19	14.01	9.41	12.39	19.42	20.06	15.06
t+20	14.07	9.39	12.53	18.73	19.83	14.91
t+21	14.54	9.51	12.65	19.34	19.36	15.08
t+22	13.98	9.65	12.74	21.24	19.43	15.41
t+23	14.36	9.87	12.86	20.08	19.56	15.35
t+24	13.89	9.93	12.94	21.89	20.41	15.81
t+25	14.27	10.22	13.09	22.13	21.88	16.32
t+26	14.09	10.12	13.22	22.30	20.91	16.13
t+27	13.98	10.29	13.28	22.51	20.73	16.16
t+28	14.01	10.35	13.34	21.13	23.82	16.53
t+29	13.78	10.52	13.37	21.31	23.07	16.41
t+30	13.75	10.67	13.41	22.91	19.58	16.07

As regards the comparison of this model with the other two reference models, we note that it is significantly more accurate than the linear regression model, as is the baseline model. However, on average, the baseline model is better than this one. Indeed, even for the 30-day prediction, we observe that the baseline model achieves nearly 3 less hospitalizations error than the dense model. Finally, the baseline model is the best among the three for all average predictions.

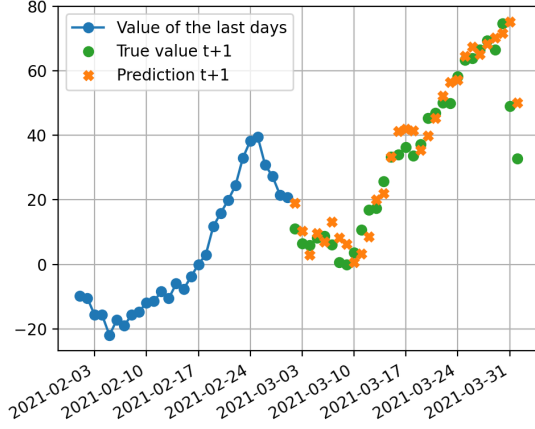


Figure 6.8: Prediction of the number of new hospitalizations in Belgium by a dense model compared to the true value

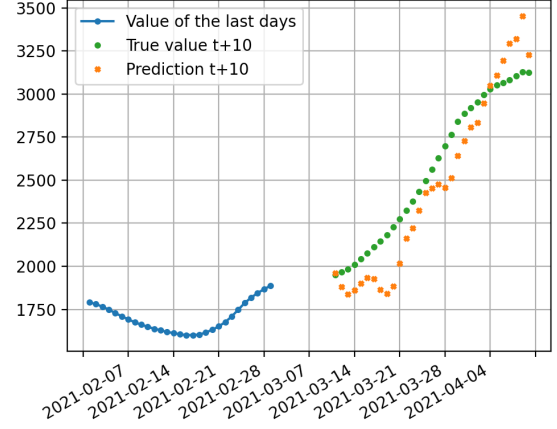


Figure 6.9: Prediction of the number of total hospitalizations in Belgium by a dense model compared to the true value

6.3.2 Encoder-decoder model

This model is the most complex of those we have built and/or used. Its parameters are described in section 5.2 and involve more than a thousand weights that need to be fixed. We evaluated the performance of this model with two different types of input: with and without *Google Trends* data. Without *Google Trends* data means that the prediction model only takes hospitalization data as input for training.

Intuitively, we thought that this complex model would be the best performing of those we tested. Unfortunately, we realized that this was not the case on all horizons of prediction. This model seems too complex to produce accurate short-term predictions. However, it is very good at producing long-term forecasts.

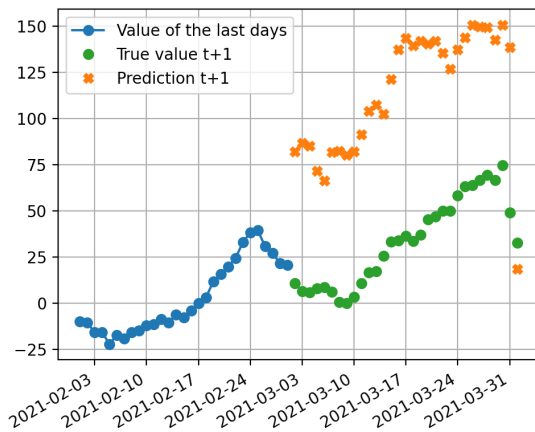


Figure 6.10: Prediction of the number of new hospitalizations in Belgium by an encoder-decoder model with *Trends* data compared to the true value

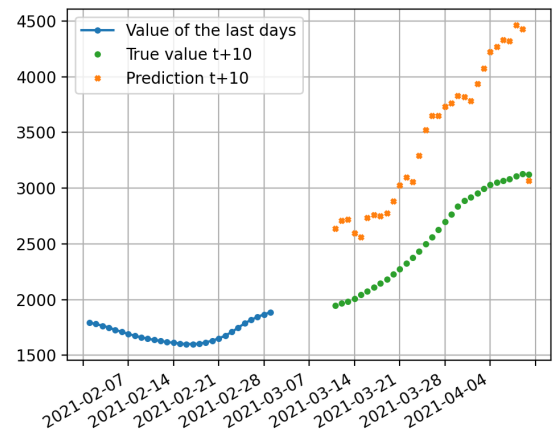


Figure 6.11: Prediction of the number of total hospitalizations in Belgium by an encoder-decoder model with *Trends* data compared to the true value

The results using hospitalization and *Google Trends* data as input of the model are

shown in Table 6.10. From those results, we see that the error is quite high on the first walks. Indeed, for a one-day prediction, the encoder-decoder model makes an error 3 times higher than the error made by the baseline model. Moreover, we can see how poor the one-day prediction is on graph 6.10. However, we are seeing something very interesting. The error of this model seems to be almost constant, whatever the prediction horizon that is considered. Thanks to this, the model is able to achieve an accurate 30-days prediction since the error is on average a little more than 11 hospitalizations at this horizon.

Table 6.10: MAE on each walk when predicting new hospitalizations for the encoder decoder, using both *Trends* and hospitalizations data, for up to 30 horizons. The mean over all walks is also reported. Boldface indicates the best performance on each row. The training dataset covered the French regions and Belgium, composed of 23 initial regions and 156 augmented regions

Horizon	Jan 2021	Feb 2021	Mar 2021	Apr 2021	May 2021	mean
t+1	18.55	6.00	9.93	11.95	6.08	10.50
t+2	17.03	5.92	10.09	12.52	6.94	10.50
t+3	16.73	6.17	10.52	12.73	6.42	10.51
t+4	15.68	6.41	10.63	12.78	5.68	10.24
t+5	14.96	6.70	10.83	12.87	5.99	10.27
t+6	14.43	7.07	11.09	13.07	6.54	10.44
t+7	14.20	7.43	11.41	13.40	6.58	10.61
t+8	14.14	7.75	11.57	13.64	6.24	10.67
t+9	14.17	7.97	11.74	13.87	5.63	10.68
t+10	14.40	8.17	11.88	14.01	5.56	10.80
t+11	14.74	8.41	12.18	14.04	5.85	11.05
t+12	15.14	8.67	12.52	13.86	5.85	11.21
t+13	15.53	8.86	12.79	13.55	5.90	11.33
t+14	15.87	9.02	12.98	13.23	5.87	11.40
t+15	16.10	9.18	13.09	12.85	5.21	11.29
t+16	16.10	9.41	13.11	12.41	5.21	11.25
t+17	15.93	9.65	13.06	11.94	5.49	11.22
t+18	15.67	9.90	12.92	11.42	5.59	11.10
t+19	15.25	10.07	12.57	10.99	6.33	11.04
t+20	14.76	10.19	12.16	10.54	6.97	10.92
t+21	14.16	10.29	11.71	10.11	7.68	10.79
t+22	13.64	10.34	11.29	9.70	8.27	10.65
t+23	13.18	10.36	10.90	9.26	8.35	10.41
t+24	12.80	10.38	10.62	8.88	8.46	10.23
t+25	12.50	10.34	10.47	8.66	8.86	10.17
t+26	12.34	10.32	10.48	8.52	9.04	10.14
t+27	12.40	10.35	10.65	8.53	8.99	10.18
t+28	12.62	10.35	11.13	8.73	9.15	10.40
t+29	12.78	10.45	11.77	9.11	9.19	10.66
t+30	12.97	10.61	12.46	9.63	10.48	11.23

On graph 6.11, we can see that the 10-day prediction curve seems to follow more or

less the same shape as the true values curve. By checking in Table 6.10, we can confirm that at a horizon of 10, this model obtains better results than those obtained by the dense model and the linear regression model. Nevertheless, the baseline model is still better than the encoder-decoder model for this horizon but this is no longer the case for the more distant horizons.

Let us now present you the results we obtain by trying to predict the number of new hospitalizations due to COVID-19 in Belgium without using the data collected via the *Google Trends* tool. We will therefore only use hospitalization data as input of the prediction model. These results are shown in Table 6.11.

In this table, we first observe that the mean absolute error that is committed by this model is strictly smaller than the one presented just before on every horizon. Nevertheless, the absolute error committed when making a one-day prediction is almost twice as large as that made by the linear regression model for the same horizon. For larger horizons, we observe that the error made by this encoder-decoder model is not as large as for a linear regression. Indeed, for a 30-day prediction, this encoder-decoder model makes a smaller absolute error of 15 hospitalizations than the linear regression. These results lead this encoder-decoder model to be the most accurate model for long-term predictions.

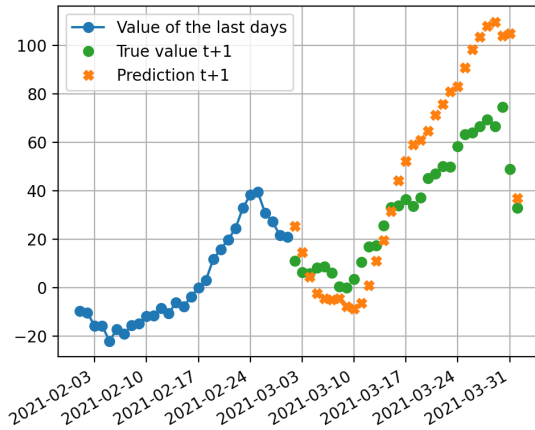


Figure 6.12: Prediction of the number of new hospitalizations in Belgium by an encoder-decoder model without *Trends* data compared to the true value

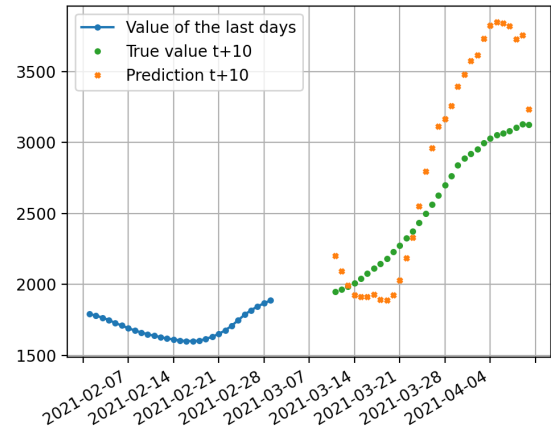


Figure 6.13: Prediction of the number of total hospitalizations in Belgium by an encoder-decoder model without *Trends* data compared to the true value

On graph 6.13, we are surprised to observe that the curve formed by the cloud of points representing the predictions does not have at all the same shape as the curve representing the true values. This prediction model seems to accentuate the extreme values of the prediction curve. The same behavior can also be observed on graph 6.12 in a less pronounced way. On this graph, we clearly observe the poor quality of the one-day prediction of this model.

It is strange to observe that the prediction curves of the two encoder-decoder models

(with and without *Google Trends* data) we have just analyzed do not resemble each other at all. We can therefore deduce that adding the *Google Trends* data as input to the prediction model disturbs the final predictions of the model.

Table 6.11: MAE on each walk when predicting new hospitalizations for the encoder decoder model, using only hospitalization data, for up to 30 horizons. The mean over all walks is also reported. Boldface indicates the best performance on each row. The training dataset covered the French regions and Belgium, composed of 23 initial regions and 156 augmented regions

Horizon	Jan 2021	Feb 2021	Mar 2021	Apr 2021	May 2021	mean
t+1	10.65	6.12	5.73	6.39	3.70	6.52
t+2	10.42	6.10	6.83	6.24	4.49	6.82
t+3	10.74	6.50	8.11	6.29	5.05	7.33
t+4	11.05	6.78	9.24	6.43	4.91	7.68
t+5	11.27	7.27	10.34	6.61	5.36	8.17
t+6	11.45	7.75	11.27	6.98	6.30	8.75
t+7	11.73	8.19	12.05	7.47	7.33	9.35
t+8	11.94	8.35	12.66	7.65	7.52	9.62
t+9	12.20	8.51	13.16	7.90	7.52	9.86
t+10	12.52	8.48	13.63	8.20	6.64	9.90
t+11	12.76	8.49	14.09	8.50	5.34	9.84
t+12	12.97	8.45	14.43	8.82	4.92	9.92
t+13	13.15	8.46	14.64	9.10	5.13	10.10
t+14	13.26	8.41	14.81	9.30	5.63	10.28
t+15	13.32	8.38	14.95	9.51	5.59	10.35
t+16	13.36	8.31	15.00	9.74	5.90	10.46
t+17	13.28	8.39	14.98	9.91	6.83	10.68
t+18	13.21	8.54	14.89	10.11	7.49	10.85
t+19	13.13	8.68	14.73	10.34	7.89	10.96
t+20	13.02	8.92	14.52	10.55	7.69	10.94
t+21	12.86	9.15	14.21	10.74	7.56	10.90
t+22	12.73	9.40	13.88	10.88	7.41	10.86
t+23	12.56	9.57	13.59	10.93	7.45	10.82
t+24	12.43	9.82	13.31	11.00	7.81	10.87
t+25	12.26	10.03	13.15	11.12	8.60	11.03
t+26	12.10	10.30	13.12	11.25	8.68	11.09
t+27	11.98	10.62	13.07	11.40	9.13	11.24
t+28	11.96	10.92	12.92	11.59	10.00	11.48
t+29	11.91	11.23	12.63	11.79	9.52	11.42
t+30	11.86	11.48	12.13	12.03	8.33	11.17

When comparing Table 6.10 with Table 6.11, we see that the encoder-decoder model was not able to use the *Google Trends* data in order to reduce the error. The error made by the model without using the *Google Trends* data for predicting 15 days ahead is almost as high as trying to predict the values for tomorrow with the *Google Trends* data. This means that the *Google Trends* data are misused by the model. Indeed, the model

gives them too much importance, which considerably reduces the quality of the prediction.

Surprised to observe that adding *Google Trends* data as an input to the prediction model degraded the results, we decided to analyze the influence of *Google Trends* data on the quality of the model’s predictions depending on the method that was used to collect them (see section 2.2.6 for more information). Therefore, we have compared the average error committed for each horizon while training the encoder-decoder model with four different types of inputs:

1. Only hospitalization data (H)
2. Hospitalization data and *Google Trends* data collected with the minimal collection method with data augmentation (H + Min. T)
3. Hospitalization data and *Google Trends* data collected with the hourly collection method with data augmentation (H + Hr. T)
4. Hospitalization data and *Google Trends* data collected with the minimal collection method without data augmentation (w/o D.A.)

From the results shown in Table 6.12, we can conclude that the data augmentation technique used was well suited for our problem as the error decreases when using it, compared to the results obtained when using only the non-augmented data for the training. Indeed, methods that use the augmented data obtain the lowest error the first 6 horizons but also for the last 11. The data augmentation method thus allows to obtain the best predictions for short or very long term predictions.

Furthermore, we also see that the results based on the minimal collection method are better than the ones using the hourly collection method. However, the data collected with the hourly method are much more stable and less noisy than those collected with the minimal collection method. This means that the *Google Trends* data has the most positive impact when they are noisy. This confirms the hypothesis put forward in this paper [52]: «*when trained with noisy inputs, LSTMs are less sensitive to data perturbations* ». Since the data collected via *Google Trends* on an hourly basis is less noisy, we observe thus that the encoder-decoder model trained with this data is more sensitive to small data perturbations. As this type of data has a limited influence on the problem at hand compared with the data directly related to the number of hospitalizations, it is best to not account for small perturbations from it.

In summary, it is therefore preferable to use an encoder-decoder model trained with hospitalization data and *Google Trends* data collected with the minimal collection method. With these inputs, the model will be able to provide very good long-term predictions and will be robust to potential data disturbances. However, if the objective is to make short-term predictions, then it is better to train the encoder-decoder model with only hospitalization data. Nevertheless, this model remains less good than the reference models when it comes to short-term predictions.

Table 6.12: MAE when predicting new hospitalizations for encoder-decoder models trained in different ways, for up to 30 horizons. Boldface indicates the best performance on each row.

Horizon	H	H + Min. T	H + Hr. T	w/o D.A.
t+1	6.52	10.50	8.27	7.77
t+2	6.82	10.50	8.87	8.41
t+3	7.33	10.51	9.14	8.49
t+4	7.68	10.24	9.34	8.48
t+5	8.17	10.27	9.63	8.71
t+6	8.75	10.44	9.94	9.03
t+7	9.35	10.61	10.22	9.25
t+8	9.62	10.67	10.40	9.33
t+9	9.86	10.68	10.54	9.41
t+10	9.90	10.80	10.89	9.57
t+11	9.84	11.05	11.34	9.69
t+12	9.92	11.21	11.68	9.79
t+13	10.10	11.33	12.03	9.90
t+14	10.28	11.40	12.29	10.02
t+15	10.35	11.29	12.42	10.09
t+16	10.46	11.25	12.67	10.24
t+17	10.68	11.22	12.89	10.46
t+18	10.85	11.10	13.01	10.68
t+19	10.96	11.04	13.16	10.87
t+20	10.94	10.92	13.25	11.07
t+21	10.90	10.79	13.50	11.48
t+22	10.86	10.65	13.68	11.69
t+23	10.82	10.41	13.68	11.61
t+24	10.87	10.23	13.72	11.67
t+25	11.03	10.17	13.82	11.82
t+26	11.09	10.14	13.91	11.90
t+27	11.24	10.18	13.91	11.88
t+28	11.48	10.40	13.99	11.90
t+29	11.42	10.66	14.10	12.01
t+30	11.17	11.23	14.32	12.49

6.3.3 Assembled model

As explained in the previous chapter, this model is composed of two different models. The first one is used to make the basic prediction using only hospitalization data. After that, the second step is to use another model based on *Google Trends* data in order to correct the basic prediction that was done. We expect the assembler model to be better than the basic prediction that has been made.

To determine whether *Google Trends* data can improve a basic prediction, we compared several combinations of models:

- A linear regression as basic model and an encoder-decoder model as adjustment

model ($LR + E-D$).

- A linear regression as basic model and a dense model as adjustment model ($LR + Dense$).
- A baseline as basic model and a dense model as adjustment model ($Baseline + Dense$).

The average results of these three model combinations will be shown in the model comparison table of the next section. For the analysis in this section, we will use the last combination: a baseline as basic model and a dense model as adjustment model. This combination was chosen for two main reasons. On the one hand, the baseline model has proved its worth in the previous sections. Indeed, this model performed very well over all horizons, much better than the linear regression model. On the other hand, the dense model was chosen as the fitting model because it is simple, quick to train and above all because it produced accurate predictions.

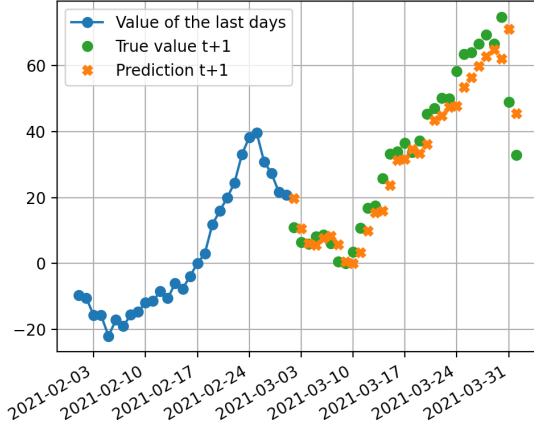


Figure 6.14: Prediction of the number of new hospitalizations in Belgium by an assembler model compared to the true value

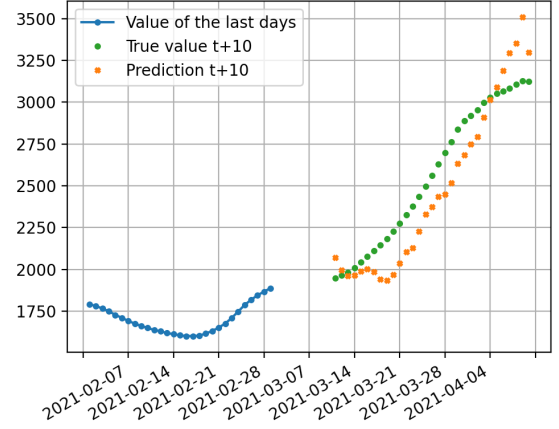


Figure 6.15: Prediction of the number of total hospitalizations in Belgium by an assembler model compared to the true value

On graphs 6.14 and 6.15, we observe that the shape of the prediction is very similar to that of the prediction produced by the baseline model, regardless of the horizon that is considered. Nevertheless, we notice that the higher the prediction horizon, the more the model seems to accentuate the extreme values of the prediction curve

In Table 6.13, we observe that, on average, the absolute error committed by the assembled model is strictly smaller than the error committed by the baseline model, except for the first horizon ($t + 1$). For the long term predictions, we observe that the assembler model achieves an absolute error that is smaller of nearly 1 hospitalizations than the baseline model. This may not seem like much of an improvement, but remember that the basic model, i.e. the baseline model, was already producing excellent results.

Thanks to this model, we can assume that the data collected on *Google Trends* plays a positive role in predicting the future number of new hospitalizations. In our case, *Google Trends* data allows us to achieve better results than only using a baseline model. Unfortunately, this performance improvement does not lead to better results than the ones provided by the encoder-decoder model.

Table 6.13: MAE on each walk when predicting new hospitalizations for the assembler model, using a baseline model assembled with a dense model, for up to 30 horizons. The mean over all walks is also reported. Boldface indicates the best performance on each row. The training dataset covered the French regions and Belgium, composed of 23 initial regions and 156 augmented regions

Horizon	Jan 2021	Feb 2021	Mar 2021	Apr 2021	May 2021	mean
t+1	2.80	2.45	2.45	2.88	3.51	2.82
t+2	4.11	3.56	3.76	4.18	5.55	4.23
t+3	5.22	4.62	4.81	5.14	7.01	5.36
t+4	6.13	5.50	5.71	6.10	8.24	6.33
t+5	6.90	6.39	6.55	6.92	9.86	7.32
t+6	7.59	7.06	7.27	7.87	11.51	8.26
t+7	8.30	7.71	7.86	8.92	12.49	9.06
t+8	8.55	7.90	8.06	9.59	12.52	9.33
t+9	8.93	7.99	8.34	10.26	12.29	9.56
t+10	9.25	8.04	8.63	11.00	10.90	9.57
t+11	9.57	8.20	8.98	11.68	9.24	9.53
t+12	9.88	8.33	9.43	12.27	9.00	9.78
t+13	10.26	8.39	9.91	12.84	9.21	10.12
t+14	10.65	8.41	10.41	13.41	9.37	10.45
t+15	11.01	8.39	10.98	13.97	9.20	10.71
t+16	11.33	8.48	11.40	14.59	8.86	10.93
t+17	11.51	8.62	11.80	15.13	9.69	11.35
t+18	11.63	8.73	12.15	15.62	10.72	11.77
t+19	11.69	8.84	12.41	16.20	10.85	12.00
t+20	11.74	9.03	12.66	16.75	10.55	12.15
t+21	11.74	9.35	12.84	17.32	8.88	12.03
t+22	11.68	9.72	12.96	17.90	7.76	12.00
t+23	11.61	10.06	13.06	18.29	8.48	12.30
t+24	11.54	10.37	13.06	18.69	9.19	12.57
t+25	11.48	10.61	13.20	19.04	9.27	12.72
t+26	11.41	11.02	13.36	19.39	9.02	12.84
t+27	11.25	11.39	13.49	19.71	9.57	13.08
t+28	11.11	11.73	13.59	19.96	10.90	13.46
t+29	10.97	12.00	13.74	20.25	10.13	13.42
t+30	10.89	12.31	13.85	20.61	7.23	12.98

6.3.4 Comparison of each model

To compare more easily the models in order to find the best one, we have computed the mean error of each walk of each model, grouped by horizon. Those results are reported in Table 6.14. In this table, we clearly observe that the model that achieves the best predictions, i.e. those with the lowest error, is the assembler model for short-term predictions and the encoder-decoder model for long-term predictions.

As far as linear regression is concerned, we observe that this model does not provide good predictions, regardless of the horizon that is considered. Indeed, the error committed by the model is important for short-term predictions as well as for long term predictions. The error increases significantly and rapidly with the horizon that is predicted. This leads us to the conclusion that the regression model is the worst of all those we tested, especially when it comes to making a long-term prediction.

We then observe that the baseline model achieves satisfactory results compared to our expectations. Indeed, this model seems to be the most efficient one on the three first horizons and achieves good results on the other horizons although it is not the best one. This is interesting because this model is very simple and requires no training to perform the prediction, unlike the dense or the encoder-decoder model.

Moreover, the third reference model that was tested seems to achieve satisfactory results even if they are not the best. Indeed, the dense model produces slightly worse predictions than those produced by the baseline model on every horizon. However, it is still much better than the linear regression model.

After that, we can compare the results obtained for the three model combinations that were tested to form the assembler model. First, we observe that the results produced by the linear regression model are improved, regardless of which adjustment model is used (encoder-decoder or dense). Nevertheless, the dense model seems to correct the initial prediction in a more pronounced way than the encoder-decoder model. Regarding the baseline model adjusted with a dense model, it has already been discussed in the previous section. It clearly improves on the initial results of the baseline model. On top of that, this assembler model is the best of all those tested on horizons 4 to 16.

Finally, the encoder-decoder model is disappointing and does not meet our expectations with regard to short-term predictions. The average absolute error made by this model is by far the highest for the first few horizons. On the other hand, the model seems to perform well to make long-term predictions. It is the best model of all those tested when it comes to making a prediction more than 16 days in advance. We believe that this model would be able to produce even better results with more data. Indeed, this model is complex, it needs to set thousands of parameters during training and the amount of data we have at the moment is not yet sufficient. Nevertheless, through data augmentation, we have been able to find alternative ways to train this model so that it produces relatively

accurate results for long-term predictions.

Table 6.14: MAE when predicting new hospitalizations for the different models, for up to 30 horizons. Boldface indicates the best performance on each row. The dataset covered the French regions and Belgium, composed of 23 initial regions and 156 augmented regions

Horizon	LR	Baseline	LR + E-D	LR + Dense	Dense	Baseline + Dense	E-D
t+1	6.16	2.79	6.00	5.79	2.89	2.82	10.50
t+2	7.10	4.21	6.82	6.73	4.51	4.23	10.50
t+3	7.84	5.36	7.51	7.40	5.65	5.36	10.51
t+4	8.47	6.38	8.08	7.97	7.01	6.33	10.24
t+5	9.11	7.41	8.60	8.60	8.00	7.32	10.27
t+6	9.72	8.38	9.12	9.17	9.28	8.26	10.44
t+7	10.26	9.22	9.58	9.62	10.38	9.06	10.61
t+8	10.65	9.50	9.94	9.92	10.92	9.33	10.67
t+9	11.06	9.78	10.26	10.23	11.55	9.56	10.68
t+10	11.58	9.80	10.63	10.49	11.68	9.57	10.80
t+11	12.22	9.77	11.10	10.87	11.61	9.53	11.05
t+12	12.94	10.04	11.68	11.42	12.14	9.78	11.21
t+13	13.55	10.39	12.18	11.90	12.82	10.12	11.33
t+14	14.17	10.72	12.71	12.38	13.60	10.45	11.40
t+15	14.89	10.98	13.30	12.91	13.34	10.71	11.29
t+16	15.65	11.23	13.92	13.57	13.35	10.93	11.25
t+17	16.24	11.65	14.41	14.09	14.52	11.35	11.22
t+18	16.85	12.08	14.90	14.63	14.89	11.77	11.10
t+19	17.51	12.35	15.41	15.11	15.06	12.00	11.04
t+20	18.28	12.51	16.01	15.71	14.91	12.15	10.92
t+21	19.29	12.47	16.88	16.56	15.08	12.03	10.79
t+22	20.13	12.44	17.57	17.25	15.41	12.00	10.65
t+23	20.75	12.74	18.05	17.70	15.35	12.30	10.41
t+24	21.36	13.03	18.54	18.18	15.81	12.57	10.23
t+25	22.06	13.22	19.11	18.71	16.32	12.72	10.17
t+26	22.88	13.34	19.81	19.40	16.13	12.84	10.14
t+27	23.53	13.59	20.33	19.89	16.16	13.08	10.18
t+28	24.01	13.98	20.69	20.18	16.53	13.46	10.40
t+29	24.93	14.01	21.47	20.98	16.41	13.42	10.66
t+30	26.24	13.63	22.66	22.08	16.07	12.98	11.23

6.4 Real-time prediction

Given the number of parameters that can be changed before running the prediction model, we decided to build a real-time prediction tool that everyone can use. This tool is easy to use as it is launched with a simple command in the terminal after making the necessary installations, even for people who are not comfortable with Python.

6.4.1 Installation

Before we start explaining how to install this application, please note that it is possible to use solutions to avoid problems related to the search quota when collecting data via *Google Trends*, as explained in section 2.2.5. If you wish to collect a large amount of data, we recommend that you follow the installation instructions in Appendix A. If you want to collect data on only a few topics, you are in the right section.

The first tool you need to download on your computer is called *Miniconda*. This will allow you to install a virtual environment with a version of Python easily on your computer. You can visit [this link](https://docs.conda.io/en/latest/miniconda.html)¹ and download the *Miniconda* executable file related to Python version 3.8 depending on your OS system. During the installation process, you have to ensure to check both checkboxes at the “Advanced Installation Options” step :

- Add Anaconda to my PATH environment variable
- Register Anaconda as my default Python 3.8

Now that *Miniconda* is installed properly on your computer, the next step is to clone the repository containing the code of the real-time prediction tool. To do this, you need to ensure that *Git* is installed on your computer. If it is the case, you can simply run the following command in a terminal :

```
$ git clone https://github.com/gerardmargaux/CLI-Covid-Thesis.git
```

The next step is to create a virtual environment that you will use for running the code. You can run the following command in a terminal :

```
$ conda create --name thesis python=3.8 --yes
```

Where "thesis" is the name that is given to the virtual environment in this example. You can obviously change this name if you want. We now need to activate the virtual environment we have just created. This can be done by running the following command :

```
$ conda activate thesis
```

Now that the virtual environment is active, you can go to the folder containing the code of the tool by running this command :

```
$ cd path/to/repository
```

The path to the repository depends obviously on the emplacement where you cloned the code.

The final step that you need to do before being able to use the tool is to install all the packages and the dependencies. You can do it easily by running the following command in the terminal :

¹<https://docs.conda.io/en/latest/miniconda.html>

```
$ pip install -e .
```

If you get an error at this step, you are probably not at the root of the repository that you cloned previously.

6.4.2 Utilization

Depending on the task you want to use, you can enter two different commands in the terminal. The first accessible command can be used to collect data about the number of hospitalizations and *Google Trends* that are available in a certain region. You can run the following command:

```
$ covid-predictor collect
```

The parameters that are set by the user before running this command are as follows :

- **collection** : This defines the type of data that should be collected. This type of collection could be minimal or an hourly/daily basis. Note that the number of queries that will be made to the *Google* server will depend very much on the collection method chosen. We advise you to use the minimal collection.
- **topics** : This represents the topics that will be collected via *Google Trends* and that will be given as input to the prediction model. Generally, these topics are chosen as terms related to the coronavirus, such as the typical symptoms of this virus.
- **european_geo** : This parameter is used to determine which region the data should be collected from. The data can be collected on the European countries or on the French regions and Belgium.

You can collect data relative to any topic on *Google Trends*. However, since the goal is to predict the number of hospitalizations due to coronavirus, it seems obvious that collecting data related to this virus will help the model to improve its prediction. That being said, be careful about the language you use when collecting data. For example, if you want to collect data related to the covid testing, entering the word "testing" will not meet your expectations. Indeed, without giving further details, the word "testing" is related to software testing and not to covid testing. In this case, you should rather collect the term "testing covid" so that there is no misunderstanding.

Note that collecting the data can be time consuming due to query quota restrictions set by *Google Trends*. Indeed, if you do too much queries to the *Google Trends* server with the same IP address, the server will stop responding after a few hundred requests. That is why we added an option to our tool that allows the user to use the *Tor* server to make requests, which means that the user's IP address will be changed every time *Google Trends* refuses to respond. However, in order to use this option, there are several additional installations to do. You can find these installations in the Appendix A.

The second available command can be used to run the prediction model in order to get the real time prediction. You can run the following command:

```
$ covid-predictor run
```

The parameters that are set by the user before running this command are as follows :

- **model_type** : This parameter defines the type of model that should be used for making the prediction. This can be a trainable or an untrainable model. If the model chosen is trainable, the user have the choice between : encoder-decoder, dense, simple auto-encoder or assembler model. On the contrary, if the model chosen is untrainable, the user have the choice between a baseline or a linear regression model.
- **target** : This describes the target that the model will try to predict. This can be the total number of hospitalizations or the number of new hospitalizations.
- **localization** : This parameter defines the localization for which the prediction should be made. For example, if we want to predict the number of hospitalizations in Belgium, the localization will be "BE".
- **european_geo** : If this parameter is *True* then the model will be trained using data relative to the european countries. On the contrary, if it is *False*, the model will be trained data relative to Belgium and French regions.
- **print_results_on_terminal** : This parameter can be set to *True* in order to print the results of the prediction on the terminal when the execution is terminated.
- **plot** : This parameter can be set to *True* in order to generate an interactive plot representing the prediction that was done for the parameters that were chosen by the user.

These parameters are global and must be specified regardless of the model the user wishes to use to make the prediction. After that, there are other parameters that should be specified depending on the type of model that was chosen. If the user wants to make a prediction using a trainable model, it has to specify the following parameters :

- **name** : This represents the name of the trainable model that should be trained among those described in Chapter 5.
- **days_to_use** : This represents the number of days that should be used by the model to make the prediction.
- **days_to_predict** : This parameter is used to determine the number of days that should be predicted by the model. We recommend using a number of days to be predicted that is smaller than or equal to the number of days that were used to deduce the prediction.
- **date_begin** : This defines the date from which the model should start training. Usually this is set to 01/02/2020 because that is when the coronavirus started to arrive in Belgium.

- **epochs** : This is the number of epochs that the model should do during the training process. It is necessary to provide a number of epochs higher than 1000 to hope to obtain a correct result. We recommend using between 3000 and 5000 epochs. However, the training may be very slow for an encoder-decoder model.
- **topics** : This represents the list of topics that should be considered by the model during the training. These topics must have been collected previously before being able to use it in the prediction.
- **scaler** : This parameters is used to determine the scaling that should be done on the data during the training. This can be a **MinMax** or a **Standard** scaler.
- **predict_one** : This parameter can be set to **True** in order to generate a unique prediction at **n_forecast** days ahead. If it is set to **False**, the prediction will be done for each day between today and today + **n_forecast** days.
- **cumsum** : This parameter can be set to **True** if we need to consider the cumulative sum of the number of new hospitalizations. It is set to **False** if the goal is to consider the absolute number of hospitalizations.
- **verbose** : This parameter can be set to **True** if the user wants to print the evolution of the training process and is set to **False** otherwise.

Note that training a complex deep learning model such as an encoder-decoder can be time consuming without using any GPU device. To avoid training models every time the user wants to make a prediction, we allow the user to choose whether or not to train the model in the case where such a model has already been trained previously. If the user decides to reuse the model that has been saved, the training process will be skipped and the prediction will be done directly.

If the goal of the user is to use a simple model that does not need a training process, it has to specify the following parameters :

- **name** : This represents the name of the untrainable model that should be used for the prediction task among those described in Chapter 5.
- **days_to_use** : This represents the number of days that should be used by the model to make the prediction.
- **days_to_predict** : This parameter is used to determine the number of days that should be predicted by the model.

With this tool, everyone can be curious and discover the prediction of the number of hospitalizations in different regions. We have tried to give as much freedom as possible to the user so that this tool can be used by most people.

In this section, we will first discuss the data collection that was explained at the beginning of this report. After that, we will consider the importance of the features collected via the *Google Trends* tool. Finally, we will discuss the results of the prediction models presented in the previous section.

7.1 Data collection

When collecting data related to the coronavirus pandemic via the *Google Trends* tool, we had the opportunity to see how unstable the data was. We have therefore considered ways of cleaning up this data as much as possible in order to obtain data that could be used by a prediction model. In section 2.2.6, we observed that the method used to collect the data via *Google Trends* has an important impact on the signals we construct. Depending on it, we can obtain data with a relatively high or low amount of noise.

We have decided to use the noisiest method to collect the data, also called minimal data collection method, because it allows us to collect data with only a small amount of requests to the *Google* server. As explained earlier in section 2.2.5, we cannot afford to send several hundreds of queries everyday as *Google* sets a quota of queries that cannot be exceeded. This was unfortunately the case with the other collection methods and using them require to introduce more resources and techniques, such as using *Tor*, adding a sleeping time between two queries and/or randomizing the requests¹. Using the minimal data collection method was later confirmed to be a rather good choice when examining Table 6.12: the forecast error of the encoder-decoder model is lower when using this method than when using the method producing the lowest amount of noise, i.e. the hourly collection method. At first, this result may appear to be somewhat counter-intuitive: why should noisy data produce better results than denoised ones? This can be explained by the fact that the encoder-decoder model is able to understand that small variations in the *Google Trends* data should not be taken into account, but only larger ones. This statement was also mentioned in this paper [52]: «*LSTM trained on noisy data becomes less susceptible to the perturbation in the data* ». As a reminder, the encoder-decoder model we use consists of several layers of LSTMs.

In section 6.3.3, we observed that a basic prediction produced by a linear regression or a baseline model could be improved by using data collected via *Google Trends* (with a minimal data collection method) through a correction factor. However, even if the minimal

¹For further details about the implementation of such methods, please refer to appendix B or the following GitHub repository <https://github.com/gerardmargaux/CLI-Covid-Thesis>

data collection method allows us to obtain better predictions than if the model was trained on hourly data, the predictions obtained are not always accurate. Moreover, the impact of the *Google Trends* data is less important on other models, such as the encoder-decoder model. For this model, we observe that the error does not always decrease when the *Google Trends* data are taken into account. Furthermore, when this is the case, the error is only slightly reduced. Our three non-exclusive hypotheses to explain this behavior are the following:

1. The architecture of the prediction model is not adapted to the problem.
2. The data collected via *Google Trends* is not reliable and biases the results of our predictions.
3. Not enough data is available to be used as input to a complex machine learning, as the crisis is quite recent.

We have done our best to provide coherent models architecture and tried several combinations of parameters and hyper-parameters for each model. Therefore, we believe that the first hypothesis is unlikely. Nevertheless, this hypothesis cannot be simply rejected, and must be mentioned. With respect to the second hypothesis, we have previously observed that the noisy *Google Trends* data allows us to obtain better predictions with the encoder-decoder model than the noise-free data. This seems counter-intuitive and leads us to believe that this hypothesis is likely. Indeed, the collected *Google Trends* data might be unreliable and the prediction models using them can overestimate their impact during the training phase, leading to a bias in the results. The last hypothesis is almost certain. Indeed, machine and deep learning techniques require a lot of data to work properly. However, since the health crisis is recent, we do not yet have much data at our disposal. To overcome the lack of data, we used data augmentation techniques to provide more training examples for the encoder-decoder model. We proved that these techniques were effective in Table 6.12. Nevertheless, we believe that our models would be able to produce even better results with more data.

Moreover, the data collected via the *Google Trends* tool is linked to the searches that are made on the *Google* search engine. However, some countries do not use this search engine or use it very little. For example, in Russia, *Google* had just over 46% of the market share in May 2019. The majority of people seem to prefer the *Yandex* search engine in this country [53]. In China, the situation is even worse as it is forbidden to use the *Google* search engine. The population seems to use *Baidu* instead [53]. Therefore, *Google Trends* data cannot be used in all countries as it does not always reflect the searches of the majority of the population. Ideally, *Google Trends* data should be available everywhere. But this means that most people in all countries must use *Google* as their search engine. This situation is not beneficial, considering that it would lead to a monopoly in this area and that *Google* is known to not always respect the privacy of its users. This remark leads us to discuss the ethical part of the data we have used.

Finally, we can ask ourselves whether the data we have used and the use we have made of it is ethical. Ethics is a key value, especially in the world of machine learning and artificial intelligence and we are aware that we must remain vigilant and sensitive to this aspect. Indeed, we know that the process of digitalization is underway and that in the future we will always have to be vigilant and discerning about the ethical aspect of the projects we work on. Concerning the work performed for this project, no sensitive data was used as *Google* had already anonymized the data before making it available. Furthermore, data was used for benevolent purposes as it allowed us to predict the number of hospitalizations due to COVID-19.

7.2 Feature importance

In section 6.2, we have found the most important features thanks to three different methods. Nevertheless, the features highlighted differ slightly from one method to another. It is therefore difficult to deduce accurately which topics are the most important ones in general. However, we note that the most important topics found through the method based on the mean decrease in impurity (section 6.2.2) and the ones found by the feature permutation method (section 6.2.3) are very similar compared to the method based on sensitivity analysis (section 6.2.1). This might be because this last method uses an encoder-decoder model to determine which features are the most important ones. In contrast, the other two methods use a *RandomForestRegressor* model in both cases to find the most important features. We can therefore deduce that the model used to generate which features are most important is primordial and has a great impact on the results.

Regardless of the method used to determine the importance of features, we noticed that the most relevant topics change over time. This is quite logical because people may be interested in different topics related to the coronavirus depending on the period. Indeed, at the beginning of the health crisis, there was very little testing being done. Therefore, very few people were interested in testing. However, when large-scale testing began, we saw a marked increase in the importance of this topic. The same behavior could be observed in relation to vaccination. Indeed, the terms related to vaccination were the most important ones from January 2021, as this is when the vaccination campaign started in Belgium.

Finally, a last criticism that can be made about the features collected via the *Google Trends* tool is that we do not exactly know what all of them cover. The topic "Fever" might be relatively simple to interpret: it should include searches related to it in several languages, as well as searches about headaches, checking for temperature and so on. However, the topic "Coronavirus disease 2019" is more difficult to interpret. As *Google's* algorithms are closed source, we do not know if this topic is solely related to the symptoms of the coronavirus, the research people are doing on the number of deaths in several countries, or even the drugs or medications currently used to fight the symptoms of the coronavirus. As this topic appears frequently as an important topic in section 6.2, we would have liked to know what it encompasses.

7.3 Prediction

The first element that we can discuss about the results related to the prediction of the number of new hospitalizations due to coronavirus is the evaluation method that was used. As a reminder, we decided to use a walk forward method. During the performance analysis, we realized that the different walks were not as easy as each other. Indeed, we noticed that the predictions were very simple on some walks and very complex on others since the model obtained sometimes a high error and sometimes a low error depending on the walk. The prediction is complex on wave periods, i.e. intervals where a peak of hospitalizations happened. Indeed, as the number of hospitalizations is very unstable during these periods, it is extremely difficult for prediction models to produce accurate predictions. That is why our method of evaluating the error of the prediction in walk forward makes sense: it allowed us to analyse the error of each model on the different walks before making an average, taking periods with or without waves into account.

Regarding the choice of target we made, our goal is to predict the new hospitalizations due to COVID-19, i.e. the variation of the number of hospitalizations instead of its total amount. As the total number of hospitalizations is a much larger value than new hospitalizations, simple models such as the baseline model will be unable to give realistic predictions for this type of target. In addition to this, the range of prediction of a trainable model is dependent on what has already been seen during the training phase. A model trained to output the total number of covid-related hospitalizations will have trouble to give a forecast exceeding the previous maximum observed, as the scaling will be done according to this extremum.

Regarding the prediction models we used, we thought that the encoder-decoder model would give the best results since it uses LSTMs. Indeed, this type of layer is relatively modern and is targeted for time series problems [54]. However, the encoder-decoder seems too complex to produce accurate short-term predictions. This means that the model fails to train properly, i.e. to set all weights in an adequate and optimal way for this kind of prediction. This situation arises when the model lacks data to train on. Despite our efforts to augment data, we realized that it would require far more data for the encoder-decoder model to become the best performer between all models and on all horizons. However, this model is still very good at producing long-term results since it is the model with the smallest error from horizon 17.

Moreover, if two different models produce similar results for predicting the number of hospitalizations due to COVID-19, it will always be preferable to use the simplest model. Indeed, a simple model is less prone to overfitting and should be able to provide better results on test sets than other models. A simpler model also has the benefit of being generally faster to train, instead of the complex encoder-decoder model requiring a GPU to be trained in a reasonable amount of time.

The prediction models we have used have been shown to be successful in predicting the number of new hospitalizations. Indeed, the best long-term prediction model is able

to provide a 30-day prediction with an absolute error of only 11 hospitalizations. These results were obtained using data related to Belgium and some French regions, and can be applied easily on those locations. However, if we want to apply the same methods on other countries, we need to take into account that the data needed to make such a realistic prediction are not always available reliably everywhere. Indeed, in some places, the reported evolution of the coronavirus crisis might be untrustworthy. This might be the case in India and China, accused by U.S officials or locals of under reporting the figures [55, 56, 57]. Even more seriously, it has been estimated that the figures announced in India and Pakistan should be multiplied by 5 or 10 [58, 59]. The lack of transparency in these regions raises questions.

We might also have tried to predict the number of deaths instead of the hospitalizations. However, it is sometimes overestimated in some countries. This might be the case in Belgium, where all the people suspected of having died from the coronavirus were counted in the number of deaths, even though no tests were carried out to verify this hypothesis [60]. This means that trying to predict the number of deaths instead of hospitalizations can also be prone to errors, and was not considered in our work.

Knowing that the coronavirus figures announced by the different countries of the world can be prone to errors, it could be interesting to make a prediction of the number of hospitalizations in these countries only on the basis of *Google Trends* data. This would allow us to detect whether the prediction we make is similar to the numbers that have been announced. If the difference between the two is significant, it is possible that the country in question is lying about the coronavirus figures. However, making a prediction of the number of hospitalizations based solely on *Google Trends* is not very accurate and the results should therefore be interpreted with caution. One should compare the error margin between the predicted and reported hospitalizations in such countries with the error margin for countries where the reporting has been considered trustworthy. If the error margin is way higher, this might indicate that the country is indeed providing false figures. However, the models only provide predictions in a scaled range, and rely on the maximum reported result in order to unscale the prediction. If the figures are under-reported, the model might fail to provide accurate results. A final note on this problem is that if we wish to know if the reported number of hospitalizations matches with the expected number of hospitalizations, we can switch to an estimation problem instead of a forecast one. Instead of predicting what will happen in the next days, we can estimate the values at time t based on observations of related signals at time t and before. By doing this, because we switch to another type of problem, other model architectures than the ones discussed here are better suited for such estimation situations.

Finally, predictions of hospitalizations on the scale of regions or countries is useful but the number of beds available for hospitalizations is spread over such area. The situation could not be alarming in some part of a region, with only a few hospitalized people. Meanwhile, in the other part of the same region, the hospitals might have a lot of hospitalized people, even needing to transport them to other locations with available

beds [61]. Even if a prediction model can depict accurately what will happen on a global scale, the situation must also be examined locally, hospital after hospital, to ensure that each one of them has enough beds available. Transport of patients is possible in order to spread the charge across the region but can be costly [62].

«It's difficult to make predictions, especially about the future »- Niels Bohr [63]

Human beings have been trying for decades to predict the future. It is therefore not surprising that temporal predictions are currently used in a wide range of domains such as weather predictions, financial market predictions, astronomy predictions, etc. However, time series forecasting is a difficult problem and the difficulty increases with the horizon we wish to predict. Indeed, making a prediction for the next day is relatively simple, but a prediction for 10, 20 or 30 days is even more complex.

The aim of this work was to create a tool capable of predicting the number of future hospitalizations happening due to COVID-19. Being able to make this prediction accurately is of the utmost importance. Indeed, it would allow hospitals to prepare for a new wave of hospitalizations or for our leaders to take decisions more quickly to avoid hospital saturation and thus save lives. In the introduction, we asked ourselves four questions and we are now in a position to answer them after having done this work.

First of all, we were able to predict the number of new hospitalizations caused by the coronavirus quite accurately for the next 30 days. Indeed, the best results that we got reach a mean absolute error of 2.79 hospitalizations when predicting the new hospitalizations for tomorrow and of 11.23 for the new hospitalizations in 30 days. In Belgium, the (rolled) new hospitalizations range from -46.86 to 74.71 during our tested periods, and our predictions of those hospitalizations are therefore off by 11.23 hospitalizations on average when predicting them 30 days ahead. These figures are based on an average on the error of our models on the prediction of the previous five months: from January to May 2021. We trained these models on data for Belgium and French regions.

Next, we tried to observe whether *Google* searches could be used to improve or correct a basic prediction. We came to the conclusion that prediction models themselves do not benefit much from the *Google Trends* data. Indeed, this data can be used to enhance a rather incorrect first prediction, such as a linear regression, but does not offer much benefices when applied to a first accurate prediction. In some specific cases, results even tend to worsen when using *Google Trends* data. This behaviour can be caused by several things: the volatile nature of the *Google Trends* data, the collection method that was used, the irreducible error when dealing with time series forecasting or even because of bad neural networks architecture. Nevertheless, as we have tried several architectures and combinations of parameters, we tend to reject this last hypothesis. It might be possible to use this data with another model architecture that was not presented here, but the

Google Trends data must be used with caution, and given less importance than the other types of data that are available. We therefore conclude that the *Google Trends* data can be used to predict the number of new hospitalizations due to COVID-19 but with a high error margin. When doing short-term predictions, it is better to ignore them and focus on the past number of hospitalizations instead. However, despite their volatile nature, they are available in most countries of the world. A model using only this type of data as input would give a prediction of the evolution of the coronavirus, rather incorrect but easy to construct and independent of the official figures provided.

After that, we asked ourselves whether all the topics were equally important in predicting the number of new hospitalizations due to COVID-19. We realized that it was not true, as some topics are more important than others. Nevertheless, it is difficult to generalize the importance of a each topic because it changes over time. For example, we observe that the searches about topics such as "Cough" or "Sore throat" had a strong positive influence on our prediction models in September 2020. However, this importance has changed over time and new topics, such as "COVID-19 testing" or "Vaccination", have emerged and proved to be more relevant later on. From this we can deduce that a good prediction system should be able to give more or less importance to a particular topic depending on the time period considered.

Finally, we collected data for many European countries in order to generalize our predictions to all these countries. Our models can therefore easily be used to make predictions outside Belgium. Yet, it is important to note that some countries lack transparency on officially published hospitalization data. Therefore, a bias can be introduced in the predictions. In addition to this, the reliability of the data related to *Google Trends* depends on the number of people using *Google*. Thus, it is important to consider that it might be less relevant for countries where *Google* is widely used as a search engine and where the population is large enough to expect accurate predictions.

For the time being, neural networks composed of LSTMs seem to be commonly used architecture for making time series predictions [44]. With regard to the data we used as input to our models, data linked to *Google Trends* has limitations and is too unstable for us to confirm that it can significantly help to improve a basic prediction. In addition, this data only represents searches that have been made on the *Google* search engine. However, some countries do not use this search engine or use it rarely, such as China, where the use of *Google* is prohibited. In order to improve the prediction model we have now, we could therefore use other open source data to rectify a basic prediction. For example, we could add open source data related to the weather, as environmental factors might influence the spread of coronavirus [64]. It would also be important to build a model that takes into account vaccination data as this is a factor that will have a non-neglectable impact on the number of hospitalizations due to COVID-19 in the future.

Bibliography

- [1] G. Derval, V. François-Lavet, and P. Schaus, “Nowcasting covid-19 hospitalizations using google trends and lstm,” in *AI4SG*, 2020.
- [2] “COVID Live Update.” <https://www.worldometers.info/coronavirus/>. [Online; accessed 24. April 2021].
- [3] “Coronavirus strongly affects Belgian economy due to international chain reaction and weakening confidence.” <https://www.kbc.com/en/economics/publications/coronavirus-strongly-affects-belgian-economy-due-to-international-chain-reaction-and-weakening-confidence.html>, Mar 2020. [Online; accessed 10. May 2021].
- [4] “Topic: Coronavirus (COVID-19) impact on the Belgian economy.” <https://www.statista.com/topics/6987/coronavirus-covid-19-impact-on-the-belgian-economy/#dossierSummary>, May 2021. [Online; accessed 10. May 2021].
- [5] admin sciensano, “COVID-19 impact,” *For a Healthy Belgium*, May 2021.
- [6] “Ce qu’il se passera si on déconfiner en mars, avril ou mai: voici comment des biostatisticiens ont calculé l’évolution de l’épidémie.” https://www.rtbef.be/info/societe/detail_no-model-is-perfect-comment-les-experts-ont-ils-calculer-l-evolution-de-l-epidemie-et-queelles-sont-les-limites-de-la-methode?id=10703819&fbclid=IwAR0aWAHLkUL3ySmkU_yWQq1XEOUjd4DE_bA8qhviD1S1FVCTXP6cyie06eg. [Online; accessed 26. May 2021].
- [7] G. Gotev, “Belgium says transparency explains high virus death toll,” *Www.euractiv*, Apr 2020.
- [8] Contributors to Wikimedia projects, “Google Flu Trends - Wikipedia.” https://en.wikipedia.org/w/index.php?title=Google_Flu_Trends&oldid=986851411, Nov 2020. [Online; accessed 10. May 2021].
- [9] “Google Trends.” <https://trends.google.fr/trends/?geo=BE>, Feb 2021. [Online; accessed 4. Feb. 2021].
- [10] “Epistat – COVID-19 Monitoring.” <https://epistat.wiv-isp.be/covid>, Feb 2021. [Online; accessed 4. Feb. 2021].
- [11] “Sciensano.” <https://www.sciensano.be/fr>, Feb 2021. [Online; accessed 4. Feb. 2021].

- [12] “Données covid-19 et Sciensano: des chercheurs se plaignent d’un manque d’accès à l’open data.” https://www.rtbef.be/info/societe/detail_open-data-et-sciensano-manque-de-coherence-entre-le-discours-officiel-et-la-realite?id=10668706, Jan 2021. [Online; accessed 4. Feb. 2021].
- [13] “Données hospitalières relatives à l’épidémie de COVID-19 - data.gouv.fr.” <https://www.data.gouv.fr/fr/datasets/donnees-hospitalieres-relatives-a-lepidemie-de-covid-19>, Feb 2021. [Online; accessed 4. Feb. 2021].
- [14] “Coronavirus (COVID-19) Hospitalizations - Statistics and Research.” <https://ourworldindata.org/covid-hospitalizations>, Mar 2021. [Online; accessed 22. Mar. 2021].
- [15] “Data on hospital and ICU admission rates and current occupancy for COVID-19.” <https://www.ecdc.europa.eu/en/publications-data/download-data-hospital-and-icu-admission-rates-and-current-occupancy-covid-19>, Mar 2021. [Online; accessed 22. Mar. 2021].
- [16] “pytrends.” <https://pypi.org/project/pytrends>, Feb 2021. [Online; accessed 4. Feb. 2021].
- [17] “COVID-19 : ce qu’il faut savoir.” <https://www.who.int/fr/emergencies/diseases/novel-coronavirus-2019/question-and-answers-hub/q-a-detail/coronavirus-disease-covid-19#:~:text=symptomes>. [Online; accessed 11. May. 2020].
- [18] “Spearman’s rank correlation coefficient.” https://en.wikipedia.org/wiki/Spearman%27s_rank_correlation_coefficient. [Online; accessed 07. June 2021].
- [19] “The Tor Project | Privacy & Freedom Online.” <https://www.torproject.org>, Apr 2021. [Online; accessed 20. May 2021].
- [20] “Garbage in, garbage out.” https://en.wikipedia.org/wiki/Garbage_in,_garbage_out. [Online; accessed 23. Apr. 2021].
- [21] “Pandas Dataframe.” <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html>. [Online; accessed 23. Apr. 2021].
- [22] “Normalization vs Standardization.” <https://www.geeksforgeeks.org/normalization-vs-standardization/>. [Online; accessed 24. Apr. 2021].
- [23] “Log Transformation: Purpose and Interpretation.” <https://medium.com/@kyawsawhtoon/log-transformation-purpose-and-interpretation-9444b4b049c9>. [Online; accessed 23. Apr. 2021].
- [24] “Recurrent Neural Networks for Time Series Forecasting: Current Status and Future Directions.” https://www.researchgate.net/publication/335599901_Recurrent_Neural_Networks_for_Time_Series_Forecasting_Current_Status_and_Future_Directions. [Online; accessed 08. June 2021].

- [25] “Data augmentation.” https://en.wikipedia.org/wiki/Data_augmentation. [Online; accessed 24. Apr. 2021].
- [26] “Artificial Neural Networks Technology.” <http://www2.psych.utoronto.ca/users/reingold/courses/ai/cache/neural4.html>. [Online; accessed 22. Apr. 2021].
- [27] F. Blayo and M. Verleysen, “Les réseaux de neurones artificiels,” 1996.
- [28] “Artificial neural network.” https://en.wikipedia.org/wiki/Artificial_neural_network#History. [Online; accessed 22. Apr. 2021].
- [29] Contributors to Wikimedia projects, “Vanishing gradient problem - Wikipedia.” https://en.wikipedia.org/w/index.php?title=Vanishing_gradient_problem&oldid=1006243644, Feb 2021. [Online; accessed 24. May 2021].
- [30] “A High-Level Introduction to LSTMs.” <https://medium.datadriveninvestor.com/a-high-level-introduction-to-lstms-34f81bfa262d>. [Online; accessed 24. Apr. 2021].
- [31] Y. Gal and Z. Ghahramani, “A theoretically grounded application of dropout in recurrent neural networks,” 2016.
- [32] “What’s the difference between a bidirectional LSTM and an LSTM ?.” <https://stackoverflow.com/questions/43035827/whats-the-difference-between-a-bidirectional-lstm-and-an-lstm>. [Online; accessed 25. Apr. 2021].
- [33] “Differences Between Word2Vec and BERT.” <https://medium.com/swlh/differences-between-word2vec-and-bert-c08a3326b5d1>. [Online; accessed 25. Apr. 2021].
- [34] “Introduction to autoencoders..” <https://www.jeremyjordan.me/autoencoders/>. [Online; accessed 25. Apr. 2021].
- [35] “Time series forecasting | TensorFlow Core.” https://www.tensorflow.org/tutorials/structured_data/time_series#baselines, Mar 2021. [Online; accessed 26. Apr. 2021].
- [36] C. Conol, “Benchmarking Methods for Time Series Forecast - Analytics Vidhya - Medium,” *Medium*, Apr 2020.
- [37] “Hyperparamètre.” <https://fr.wikipedia.org/wiki/Hyperparamètre>. [Online; accessed 26. Apr. 2021].
- [38] “Hyperparameter Optimization for Keras.” <https://github.com/autonomio/talos>. [Online; accessed 26. Apr. 2021].
- [39] “Une plate-forme Open Source de bout en bout dédiée au machine learning.” <https://www.tensorflow.org>. [Online; accessed 24. May. 2021].

- [40] “How To Backtest Machine Learning Models for Time Series Forecasting.” <https://machinelearningmastery.com/backtest-machine-learning-models-time-series-forecasting>, Aug 2019. [Online; accessed 23. Apr. 2021].
- [41] I. Kaastra and M. Boyd, “Designing a neural network for forecasting financial and economic time series,” *Neurocomputing*, vol. 10, no. 3, pp. 215–236, 1996. Financial Applications, Part II.
- [42] A. Walker, C. Hopkins, and P. Surda, “Use of google trends to investigate loss-of-smell-related searches during the covid-19 outbreak,” in *International forum of allergy & rhinology*, vol. 10, pp. 839–847, Wiley Online Library, 2020.
- [43] M. T. Ribeiro, S. Singh, and C. Guestrin, “Model-agnostic interpretability of machine learning,” 2016.
- [44] “Deep Learning for Time Series Forecasting: A Survey.” <https://www.liebertpub.com/doi/full/10.1089/big.2020.0159>. [Online; accessed 25. Apr. 2021].
- [45] “Number of cases and tests each day.” <https://www.covidata.be/cases/testing>. [Online; accessed 06. June 2021].
- [46] “Campagne de vaccination contre la Covid-19 en France.” https://fr.wikipedia.org/wiki/Campagne_de_vaccination_contre_la_Covid-19_en_France. [Online; accessed 06. June 2021].
- [47] “sklearn.ensemble.RandomForestRegressor — scikit-learn 0.24.2 documentation.” <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html>, May 2021. [Online; accessed 24. May 2021].
- [48] Scikit-learn, “Feature importance based on mean decrease in impurity.” https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html. [Online; accessed 21. May 2021].
- [49] “Conférence interministérielle Santé publique du 26 août 2020.” <https://www.health.belgium.be/fr/news/conference-interministerielle-sante-publique-du-26-aout-2020>. [Online; accessed 04. June 2021].
- [50] “CONFÉRENCE INTERMINISTÉRIELLE SANTÉ PUBLIQUE - RÉUNION PLÉNIÈRE DU 19 OCTOBRE 2020 - COMMUNICATION.” <https://www.info-coronavirus.be/fr/news/politique-despistage-adaptee/>. [Online; accessed 04. June 2021].
- [51] Scikit-learn, “Feature importance based on feature permutation.” https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html. [Online; accessed 21. May 2021].
- [52] K. Yeo, “Short note on the behavior of recurrent neural network for noisy dynamical system,” *arXiv preprint arXiv:1904.05158*, 2019.

- [53] “Parts de Marché des Moteurs de Recherche dans le monde.” https://www.opinionact.com/ressources/seo-content-marketing/parts-de-marche-des-moteurs-de-recherche-dans-le-monde#0.1_europe. [Online; accessed 01. June 2021].
- [54] Contributors to Wikimedia projects, “Long short-term memory - Wikipedia.” https://en.wikipedia.org/w/index.php?title=Long_short-term_memory&oldid=1027697866, Jun 2021. [Online; accessed 12. Jun. 2021].
- [55] S. Biswas, “India’s Covid crisis: The newsroom counting the uncounted deaths,” *BBC News*, May 2021.
- [56] C. K. Johnson, “Why India’s coronavirus cases, deaths are vastly undercounted,” *Los Angeles Times*, May 2021.
- [57] N. Wadhams, J. Jacobs, and Bloomberg, “China intentionally under-reported total coronavirus cases and deaths, U.S. intelligence says,” *Fortune*, Apr 2020.
- [58] “Le grand trafic des chiffres sur la mortalité du coronavirus.” <https://www.franceinter.fr/emissions/le-monde-a-l-envers/le-monde-a-l-envers-10-juin-2020>. [Online; accessed 25. May 2021].
- [59] “Inde : pourquoi le nombre de morts est certainement très sous-estimé.” <https://www.leprogres.fr/sante/2021/04/26/inde-pourquoi-le-nombre-de-morts-est-certainement-tres-sous-estime>. [Online; accessed 25. May 2021].
- [60] “Trop "transparente", la Belgique va revoir sa façon de compter les morts du Covid.” https://www.rtf.be/info/societe/detail_bientot-une-meilleure-comptabilisation-des-deces-causes-par-le-coronavirus?id=10483513. [Online; accessed 25. May 2021].
- [61] P. Dejong, “Dutch hospital airlifts patients to Germany amid virus surge,” *AP NEWS*, Oct 2020.
- [62] S. Kliff, “A \$52,112 Air Ambulance Ride: Coronavirus Patients Battle Surprise Bills,” *N.Y. Times*, Oct 2020.
- [63] “It’s Difficult to Make Predictions, Especially About the Future.” <https://quoteinvestigator.com/tag/niels-bohr/>. [Online; accessed 25. May 2021].
- [64] S. Bhattacharjee, “Statistical investigation of relationship between spread of coronavirus disease (covid-19) and environmental factors based on study of four mostly affected places of china and five mostly affected places of italy,” 2020.
- [65] “25 Years of IIF Time Series Forecasting: A Selective Review (Working paper).” <https://www.econstor.eu/bitstream/10419/86645/1/05-068.pdf>, 2005. [Online; accessed 22. Apr. 2021].

- [66] “Tableau de bord de données haute fréquence (Italie, Espagne, Royaume-Uni, États-Unis et Japon).” <https://www.banque-france.fr/statistiques/chiffres-clefs-france-et-etranger/tableau-de-bord/tableau-de-bord-de-donnees-haute-frequence-italie-espagne-royaume-uni-etats-unis-et-japon>. [Online; accessed 01. June 2021].
- [67] “Docker + MiniConda = A Perfect Match.” <http://pythonwise.blogspot.com/2015/04/docker-miniconda-perfect-match.html>. [Online; accessed 08. June 2021].
- [68] “A step-by-step guide how to use Python with Tor and Privoxy.” <https://gist.github.com/DusanMadar/8d11026b7ce0bce6a67f7dd87b999f6b>. [Online; accessed 08. April 2021].
- [69] 3Blue1Brown, “Backpropagation calculus | Deep learning, chapter 4.” https://www.youtube.com/watch?v=tIeHLnjs5U8&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi&index=5, Nov 2017. [Online; accessed 26. Apr. 2021].
- [70] 3Blue1Brown, “What is backpropagation really doing? | Deep learning, chapter 3.” https://www.youtube.com/watch?v=Ilg3gGewQ5U&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi&index=3, Nov 2017. [Online; accessed 26. Apr. 2021].
- [71] mjljar, “Random Forest Feature Importance Computed in 3 Ways with Python.” <https://mljar.com/blog/feature-importance-in-random-forest/>. [Online; accessed 21. May 2021].
- [72] “COVID-19 | Diagnostic.” <https://www.ssmg.be/covid-19-diagnostic>, May 2021. [Online; accessed 25. May 2021].
- [73] S. Abrams, J. Wambua, E. Santermans, L. Willem, E. Kuylen, P. Coletti, P. Libin, C. Faes, O. Petrof, S. A. Herzog, P. Beutels, and N. Hens, “Modeling the early phase of the belgian covid-19 epidemic using a stochastic compartmental model and studying its implied future trajectories,” *medRxiv*, 2020.

Appendices

In this chapter, we will see together how to set up the facilities that allow us to set up more reliable data collection by changing our IP address via *Tor* when the *Google* server decides to stop responding to our requests.

A.1 Preparation of the virtual environment

If you are on the Ubuntu operating system, you can jump straight to the next section. If you are not, please follow the instructions below carefully. You first have to install *Docker* on your computer. To do this, you can follow the installation instructions in the following link: <https://docs.docker.com/desktop/>. Once the *Docker* application is installed, you can open it to start the service. You are now ready to use *Docker* to launch an Ubuntu virtual environment. Then, you just have to run this command in your bash terminal:

```
$ docker run -it ubuntu:18.04 bash
```

Now that the virtual environment is created, it is necessary to install a version of Python. We will do this by installing Miniconda by running the following instructions on the terminal [67]:

```
# apt-get update
# apt-get install -y curl
# curl -LO http://repo.continuum.io/miniconda/Miniconda-latest-Linux-x86_64.sh
# bash Miniconda-latest-Linux-x86_64.sh -p /miniconda -b
# rm Miniconda-latest-Linux-x86_64.sh
# PATH=/miniconda/bin:${PATH}
# conda update -y conda
```

You will also need the *Git* package to be able to clone the repository where the code is located afterwards. You can run the following command in the terminal:

```
# apt-get -y install git
```

Now that *Miniconda* is installed properly on your docker environment, the next step is to clone the repository containing the code of the real-time prediction tool. You can simply run the following command in the terminal :

```
# git clone https://github.com/gerardmargaux/CLI-Covid-Thesis.git
```

The next step is to create a virtual environment that you will use for running the code. You can run the following command in a terminal :

```
# conda create --name thesis python=3.8 --yes
```

Where "thesis" is the name that is given to the virtual environment in this example. You can obviously change this name if you want. We now need to activate the virtual environment we have just created. This can be done by running the following command :

```
# source activate thesis
```

Now that the virtual environment is active, you can go to the folder containing the code of the tool by running this command :

```
# cd CLI-Covid-Thesis
```

The next step is to install all the packages and dependencies related to the application in order to be able to use it. You can do it easily by running the following command in the terminal:

```
# pip install -e .
```

You still have 2 packages to install manually so that your environment is ready to be used properly. You can run the following commands:

```
# pip install psocks
# pip install toripchanger
```

You are now ready to configure the environment in order to be able to use *Tor* properly.

A.2 Configuration of the virtual environment

The following explanations are based on this guide [68]. They have been applied to the Ubuntu 18.04 operating system. Let us begin with the installation of *Tor* before starting its service via the command line by running these different commands:

```
# apt install -y tor
# echo "ControlPort 9051" >> /etc/tor/torrc
# echo HashedControlPassword $(tor --hash-password "my password" |
tail -n 1) >> /etc/tor/torrc
# service tor start
```

The next step is to install Privoxy. You can run these different commands:

```
# apt install -y privoxy
# echo "forward-socks5t / 127.0.0.1:9050 ." >> /etc/privoxy/config
# sed -i "s/.*\[::1\]:8118/# &/" /etc/privoxy/config
# service privoxy start
```

Once the installation is complete, you can return to section 6.4.2 and collect data through *Tor*.

B

Class Diagram of the *Google Trends* collection system

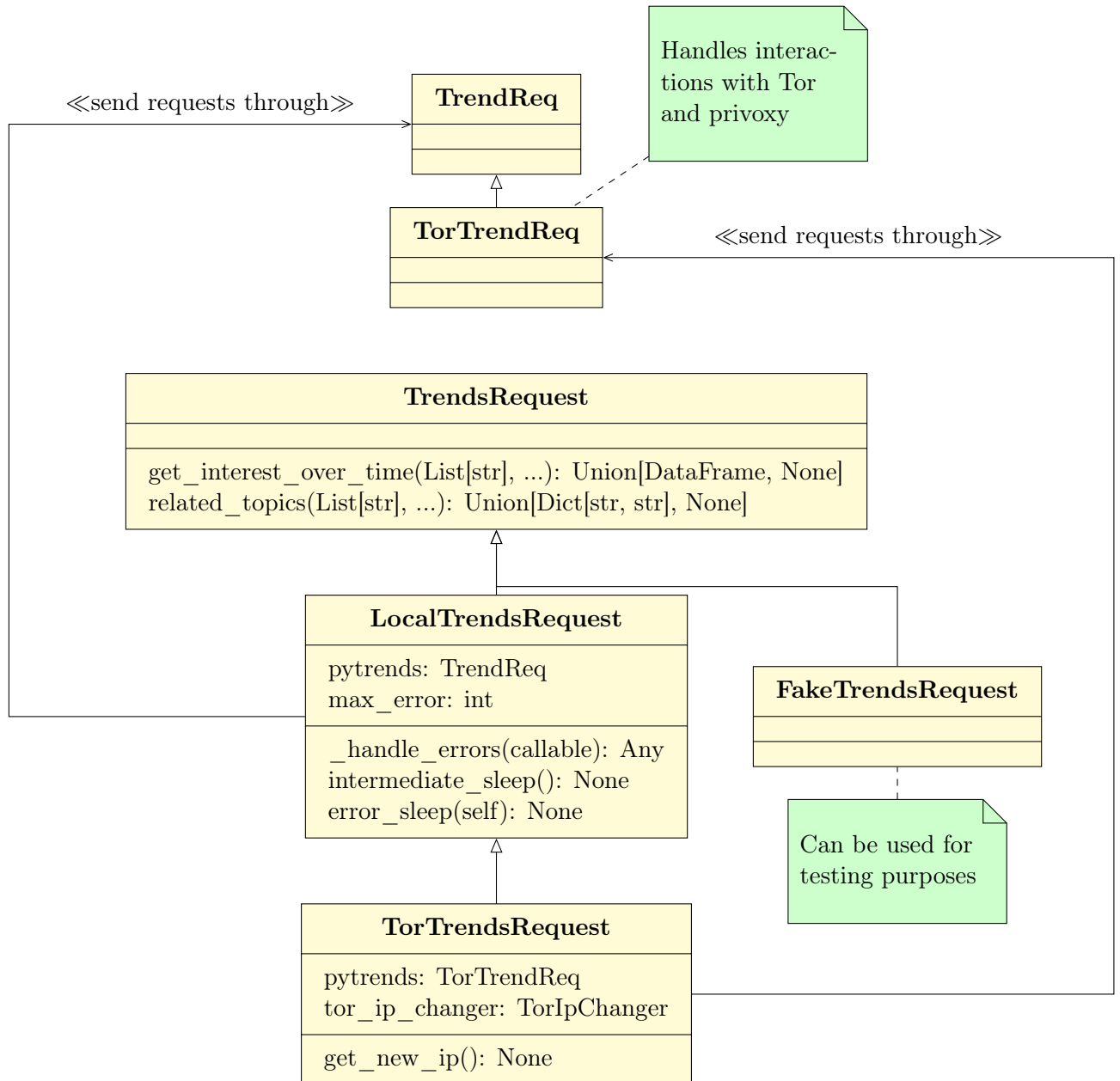


Figure B.1: Class diagram of the interactions through `pytrends.requests.TrendReq` and Tor. Only the most relevant variables and methods are shown

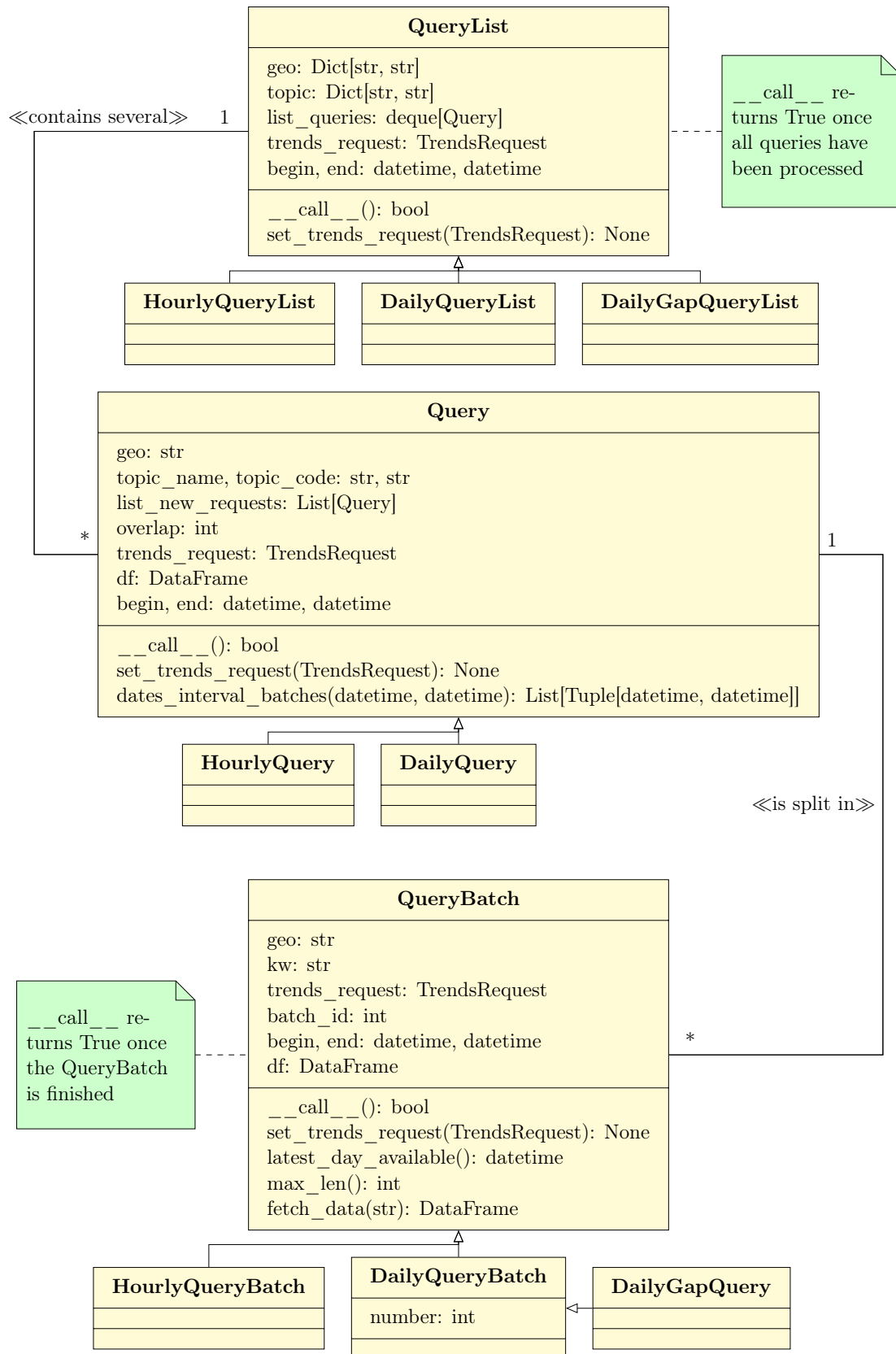


Figure B.2: Class diagram of the handling of queries over multiple locations and multiple topics. A **QueryList** contains multiple **Query** instances, each of them being split in several **QueryBatch** instances. Only the most relevant variables and methods are shown

C.1 Linear regression model

Table C.1: MSE on each walk when predicting new hospitalizations for the linear regression model, for up to 30 horizons. The mean over all walks is also reported. Boldface indicates the best performance on each row. The training dataset covered the French regions and Belgium, composed of 23 initial regions and 156 augmented regions

Horizon	Jan 2021	Feb 2021	Mar 2021	Apr 2021	May 2021	mean
t+1	80.43	158.05	76.73	111.78	44.62	94.32
t+2	102.71	205.80	101.96	144.33	61.52	123.26
t+3	123.30	258.80	127.20	181.04	68.71	151.81
t+4	144.70	315.61	149.60	224.39	69.73	180.81
t+5	165.91	376.80	170.78	271.57	80.09	213.03
t+6	188.81	440.92	189.83	321.63	86.51	245.54
t+7	212.77	502.53	209.21	370.54	84.02	275.81
t+8	239.76	557.42	224.28	423.15	80.41	305.00
t+9	271.49	604.45	240.25	481.92	78.55	335.33
t+10	312.84	641.17	259.03	546.06	85.69	368.96
t+11	350.40	679.64	293.17	602.83	121.45	409.50
t+12	393.03	712.79	343.44	652.74	153.42	451.08
t+13	433.91	742.39	398.98	709.11	163.34	489.55
t+14	475.64	767.51	460.51	771.62	177.78	530.61
t+15	512.24	795.87	528.83	837.49	236.90	582.27
t+16	546.04	829.43	591.50	916.24	311.64	638.97
t+17	572.01	873.83	656.21	1004.76	349.86	691.33
t+18	594.60	923.75	723.76	1098.33	384.68	745.02
t+19	614.62	977.96	788.19	1203.93	420.29	801.00
t+20	631.34	1041.35	860.94	1309.99	497.71	868.27
t+21	641.08	1118.49	937.42	1419.37	695.49	962.37
t+22	651.61	1199.66	1019.57	1538.12	799.65	1041.72
t+23	661.64	1287.96	1114.84	1661.34	777.60	1100.68
t+24	673.60	1388.87	1216.70	1777.76	784.64	1168.31
t+25	690.24	1486.80	1331.49	1895.40	837.62	1248.31
t+26	708.62	1588.98	1450.95	2015.05	959.59	1344.64
t+27	732.02	1694.85	1569.37	2147.71	997.37	1428.26
t+28	758.46	1801.62	1698.71	2281.33	954.91	1499.00
t+29	787.46	1909.63	1837.06	2410.68	1157.53	1620.47
t+30	816.83	2022.43	1976.63	2544.28	1683.61	1808.76

C.2 Baseline model

Table C.2: MSE on each walk when predicting new hospitalizations for the baseline model, for up to 30 horizons. The mean over all walks is also reported. Boldface indicates the best performance on each row. The training dataset covered the French regions and Belgium, composed of 23 initial regions and 156 augmented regions

Horizon	Jan 2021	Feb 2021	Mar 2021	Apr 2021	May 2021	mean
t+1	15.90	13.94	12.64	18.83	22.86	16.83
t+2	31.98	32.77	31.09	41.87	56.28	38.80
t+3	49.58	61.64	52.09	65.04	86.02	62.87
t+4	67.87	91.86	73.84	92.30	120.26	89.23
t+5	86.28	129.17	99.15	120.68	177.54	122.56
t+6	107.51	164.95	124.78	158.08	233.37	157.74
t+7	133.62	201.86	149.10	200.50	273.39	191.69
t+8	152.17	222.83	161.19	230.63	293.69	212.10
t+9	175.64	236.76	173.36	265.19	314.00	232.99
t+10	202.74	239.62	186.82	306.88	278.92	243.00
t+11	227.52	245.58	208.80	348.54	202.68	246.62
t+12	254.69	245.68	236.43	384.67	173.69	259.03
t+13	278.46	245.78	265.06	421.12	185.14	279.11
t+14	300.97	241.80	297.54	461.57	199.88	300.35
t+15	317.16	241.42	334.16	506.31	168.73	313.56
t+16	332.18	243.40	365.38	557.44	140.45	327.77
t+17	339.59	252.90	393.99	607.39	157.96	350.36
t+18	343.36	264.39	418.59	657.40	177.15	372.18
t+19	346.81	277.05	441.16	712.17	191.57	393.75
t+20	348.26	292.27	465.53	762.74	186.15	410.99
t+21	345.71	313.54	487.04	810.73	135.76	418.55
t+22	343.84	333.76	505.95	857.88	115.10	431.31
t+23	339.74	353.83	527.97	898.27	146.77	453.31
t+24	336.80	377.49	550.86	937.77	174.24	475.43
t+25	335.28	394.58	581.86	976.67	192.38	496.16
t+26	330.97	411.64	609.17	1014.82	184.46	510.21
t+27	326.49	430.09	630.92	1059.91	209.05	531.29
t+28	321.34	446.58	654.93	1105.43	277.67	561.19
t+29	313.99	463.57	681.38	1151.33	239.42	569.94
t+30	303.88	484.72	706.28	1199.99	114.86	561.94

C.3 Dense model

Table C.3: MSE on each walk when predicting new hospitalizations for the dense model, for up to 30 horizons. The mean over all walks is also reported. Boldface indicates the best performance on each row. The training dataset covered the French regions and Belgium, composed of 23 initial regions and 156 augmented regions

Horizon	Jan 2021	Feb 2021	Mar 2021	Apr 2021	May 2021	mean
t+1	18.80	15.42	13.16	20.54	15.81	16.74
t+2	34.23	37.26	33.87	40.19	72.78	43.66
t+3	58.58	58.63	51.31	78.84	102.37	69.95
t+4	88.84	91.10	77.65	92.81	219.72	114.03
t+5	115.11	131.00	110.77	161.63	257.73	155.25
t+6	146.87	198.92	138.75	228.38	378.02	218.19
t+7	179.47	228.46	171.63	247.05	559.98	277.32
t+8	212.21	247.04	190.16	299.37	705.02	330.76
t+9	271.67	279.31	214.60	364.05	858.83	397.69
t+10	320.97	278.94	221.59	526.93	818.04	433.29
t+11	341.42	256.90	239.12	530.33	794.79	432.51
t+12	399.52	279.49	265.61	721.94	803.26	493.97
t+13	495.39	273.80	285.47	714.47	1120.09	577.84
t+14	554.85	268.20	300.71	948.94	1306.39	675.82
t+15	540.68	241.49	330.40	1045.97	1113.81	654.47
t+16	581.21	244.28	352.50	1044.17	1094.28	663.29
t+17	672.53	254.14	372.49	1140.21	1485.56	784.99
t+18	700.07	277.30	389.81	1403.16	1394.17	832.90
t+19	658.14	282.72	412.55	1510.28	1490.09	870.76
t+20	673.87	266.33	431.70	1436.09	1540.66	869.73
t+21	743.42	271.62	454.91	1530.84	1604.65	921.09
t+22	689.27	290.74	470.63	1782.16	1677.83	982.12
t+23	755.52	286.17	479.49	1645.20	1845.20	1002.31
t+24	700.20	297.13	497.52	1897.41	2003.92	1079.24
t+25	769.07	294.35	532.61	1950.78	2199.52	1149.27
t+26	762.67	293.59	548.49	2000.90	2133.22	1147.77
t+27	763.36	295.84	557.05	2051.94	2121.66	1157.97
t+28	767.34	296.77	538.09	1894.43	2512.26	1201.78
t+29	753.53	304.62	546.21	1929.16	2419.78	1190.66
t+30	748.34	313.38	593.71	2148.49	1803.56	1121.50

C.4 Encoder-decoder model

Table C.4: MSE on each walk when predicting new hospitalizations for the encoder decoder, using both *Trends* and hospitalizations data, for up to 30 horizons. The mean over all walks is also reported. Boldface indicates the best performance on each row. The training dataset covered the French regions and Belgium, composed of 23 initial regions and 156 augmented regions

Horizon	Jan 2021	Feb 2021	Mar 2021	Apr 2021	May 2021	mean
t+1	835.22	98.04	410.63	736.93	109.02	437.97
t+2	659.94	89.59	425.21	760.05	121.36	411.23
t+3	622.42	93.23	474.74	751.48	95.76	407.52
t+4	527.34	100.72	478.97	724.67	81.15	382.57
t+5	468.48	106.19	494.70	696.87	95.35	372.32
t+6	430.01	115.38	518.86	678.91	96.12	367.86
t+7	420.90	126.92	552.36	670.61	89.35	372.03
t+8	435.32	137.84	578.41	658.33	81.39	378.26
t+9	468.34	146.00	597.18	646.63	71.45	385.92
t+10	517.09	151.45	614.76	633.81	65.45	396.51
t+11	566.68	157.66	651.95	615.97	62.16	410.89
t+12	619.88	164.11	701.31	585.27	61.31	426.38
t+13	669.00	170.13	744.22	545.57	64.56	438.70
t+14	711.82	176.86	778.28	506.62	72.51	449.22
t+15	736.12	185.72	806.21	469.99	81.87	455.98
t+16	744.12	198.85	818.26	433.49	115.02	461.95
t+17	726.18	213.48	812.08	391.07	138.14	456.19
t+18	694.12	228.83	785.08	351.29	145.16	440.89
t+19	652.33	242.27	736.19	315.10	182.05	425.59
t+20	603.56	254.98	680.16	282.36	204.92	405.20
t+21	554.26	268.32	617.18	253.22	273.80	393.36
t+22	514.21	277.34	555.84	226.89	345.83	384.02
t+23	485.49	282.77	507.35	203.40	340.20	363.84
t+24	467.57	288.75	475.81	188.97	332.84	350.79
t+25	461.17	289.95	462.39	182.42	354.83	350.15
t+26	464.31	289.85	467.22	184.67	387.42	358.69
t+27	474.73	290.73	490.08	197.84	404.51	371.58
t+28	491.74	290.33	534.70	222.54	395.99	387.06
t+29	512.29	293.01	599.29	264.74	389.08	411.68
t+30	535.70	297.82	680.68	328.03	488.88	466.22

C.5 Assembled model

Table C.5: MSE on each walk when predicting new hospitalizations for the assembler model, using a baseline model assembled with a dense model, for up to 30 horizons. The mean over all walks is also reported. Boldface indicates the best performance on each row. The training dataset covered the French regions and Belgium, composed of 23 initial regions and 156 augmented regions

Horizon	Jan 2021	Feb 2021	Mar 2021	Apr 2021	May 2021	mean
t+1	16.49	13.51	13.26	18.14	23.47	16.97
t+2	33.08	30.85	32.22	39.82	57.32	38.66
t+3	51.20	56.85	53.28	61.15	89.08	62.32
t+4	69.15	83.25	74.51	86.49	125.37	87.75
t+5	86.42	115.95	98.99	112.20	185.35	119.78
t+6	105.43	146.70	123.70	146.18	247.84	153.97
t+7	129.41	178.42	146.24	186.74	289.52	186.07
t+8	145.62	194.66	156.80	216.59	316.76	206.09
t+9	166.07	204.79	167.75	250.69	334.63	224.79
t+10	190.03	204.73	180.04	291.47	302.17	233.69
t+11	210.33	208.30	199.18	332.62	227.59	235.60
t+12	232.73	206.43	223.02	368.26	196.17	245.32
t+13	251.33	205.12	247.79	403.55	212.26	264.01
t+14	270.30	200.80	275.95	442.56	231.40	284.20
t+15	282.94	199.89	307.19	485.09	199.00	294.82
t+16	295.58	201.27	333.05	533.41	169.36	306.54
t+17	299.40	209.68	357.23	577.97	188.86	326.63
t+18	298.44	220.06	376.55	622.42	213.02	346.10
t+19	297.47	231.98	394.47	672.58	228.78	365.05
t+20	295.33	246.49	413.54	719.14	224.28	379.76
t+21	290.31	267.26	429.28	763.07	165.58	383.10
t+22	285.68	287.08	443.27	805.75	143.18	392.99
t+23	279.21	307.01	459.68	840.17	181.58	413.53
t+24	273.51	330.53	475.67	875.89	215.73	434.27
t+25	268.92	349.29	499.19	908.82	239.29	453.10
t+26	262.44	367.37	519.08	942.39	227.26	463.71
t+27	255.40	387.32	533.95	982.25	260.76	483.94
t+28	247.85	405.92	549.78	1021.09	339.35	512.80
t+29	239.47	424.48	568.32	1061.01	296.99	518.05
t+30	230.99	445.90	585.55	1107.63	151.67	504.35

C.6 Comparison of each model

Table C.6: MSE when predicting new hospitalizations for the different models, for up to 30 horizons. Boldface indicates the best performance on each row. The dataset covered the French regions and Belgium, composed of 23 initial regions and 156 augmented regions

Horizon	E-D	Baseline	Dense	L.R.	Baseline + Dense
t+1	437.97	16.83	16.74	94.32	16.97
t+2	411.23	38.80	43.66	123.26	38.66
t+3	407.52	62.87	69.95	151.81	62.32
t+4	382.57	89.23	114.03	180.81	87.75
t+5	372.32	122.56	155.25	213.03	119.78
t+6	367.86	157.74	218.19	245.54	153.97
t+7	372.03	191.69	277.32	275.81	186.07
t+8	378.26	212.10	330.76	305.00	206.09
t+9	385.92	232.99	397.69	335.33	224.79
t+10	396.51	243.00	433.29	368.96	233.69
t+11	410.89	246.62	432.51	409.50	235.60
t+12	426.38	259.03	493.97	451.08	245.32
t+13	438.70	279.11	577.84	489.55	264.01
t+14	449.22	300.35	675.82	530.61	284.20
t+15	455.98	313.56	654.47	582.27	294.82
t+16	461.95	327.77	663.29	638.97	306.54
t+17	456.19	350.36	784.99	691.33	326.63
t+18	440.89	372.18	832.90	745.02	346.10
t+19	425.59	393.75	870.76	801.00	365.05
t+20	405.20	410.99	869.73	868.27	379.76
t+21	393.36	418.55	921.09	962.37	383.10
t+22	384.02	431.31	982.12	1041.72	392.99
t+23	363.84	453.31	1002.31	1100.68	413.53
t+24	350.79	475.43	1079.24	1168.31	434.27
t+25	350.15	496.16	1149.27	1248.31	453.10
t+26	358.69	510.21	1147.77	1344.64	463.71
t+27	371.58	531.29	1157.97	1428.26	483.94
t+28	387.06	561.19	1201.78	1499.00	512.80
t+29	411.68	569.94	1190.66	1620.47	518.05
t+30	466.22	561.94	1121.50	1808.76	504.35

D

Performance analysis based on European data

D.1 Linear regression model

Table D.1: MAE on each walk when predicting new hospitalizations for the linear regression model, for up to 30 horizons. The mean over all walks is also reported. Boldface indicates the best performance on each row. The training dataset covered the European countries, composed of 23 initial regions and 61 augmented regions

Horizon	Dec 2020	Jan 2021	Feb 2021	Mar 2021	Apr 2021	mean
t+1	53.09	52.59	57.28	30.60	35.90	45.89
t+2	60.86	61.73	66.07	35.61	41.45	53.14
t+3	68.19	71.39	74.72	40.60	47.21	60.42
t+4	75.29	81.24	83.46	45.71	53.48	67.84
t+5	82.46	91.20	91.86	50.38	60.77	75.33
t+6	89.63	100.78	100.88	55.07	68.12	82.90
t+7	96.03	110.78	109.58	60.04	75.44	90.37
t+8	101.89	121.08	118.20	65.14	82.86	97.83
t+9	108.34	131.23	126.80	70.58	90.70	105.53
t+10	114.81	141.73	135.22	76.05	98.57	113.28
t+11	121.59	152.06	143.77	82.31	105.79	121.10
t+12	128.37	162.46	152.40	89.32	112.39	128.99
t+13	135.10	172.62	160.85	96.78	118.77	136.83
t+14	142.10	182.64	169.38	104.33	125.20	144.73
t+15	149.38	192.41	177.72	112.06	131.22	152.56
t+16	157.00	202.01	186.25	120.09	137.32	160.53
t+17	165.27	211.12	194.73	128.07	143.17	168.47
t+18	174.28	220.03	202.66	136.34	149.26	176.51
t+19	183.70	228.43	210.64	144.68	155.60	184.61
t+20	193.10	236.67	218.58	153.32	162.21	192.77
t+21	203.10	244.56	226.94	162.25	168.73	201.12
t+22	213.54	252.42	235.56	171.10	175.07	209.54
t+23	224.09	260.04	243.86	180.00	181.49	217.90
t+24	234.06	267.86	251.97	189.46	187.79	226.23
t+25	243.35	275.69	260.10	198.97	194.09	234.44
t+26	252.08	284.09	268.06	208.45	200.48	242.63
t+27	260.60	292.01	275.82	217.95	206.69	250.62
t+28	268.61	299.83	283.54	227.33	212.49	258.36
t+29	276.63	307.08	291.27	236.79	217.79	265.91
t+30	284.56	313.98	299.39	246.33	223.44	273.54

D.2 Baseline model

Table D.2: MAE on each walk when predicting new hospitalizations for the baseline model, for up to 30 horizons. The mean over all walks is also reported. Boldface indicates the best performance on each row. The training dataset covered the European countries, composed of 23 initial regions and 61 augmented regions

Horizon	Dec 2020	Jan 2021	Feb 2021	Mar 2021	Apr 2021	mean
t+1	12.27	14.55	10.15	9.39	12.37	11.75
t+2	19.12	24.16	16.88	15.48	20.39	19.21
t+3	25.12	33.53	23.68	20.92	26.99	26.05
t+4	31.03	42.37	29.90	26.86	33.11	32.65
t+5	36.57	51.21	36.11	32.20	39.17	39.05
t+6	42.49	59.55	41.84	37.60	45.87	45.47
t+7	47.72	68.01	47.56	42.67	52.36	51.66
t+8	52.07	75.42	52.46	47.39	57.40	56.95
t+9	56.58	82.88	57.32	51.92	62.35	62.21
t+10	60.80	90.18	62.07	56.68	67.12	67.37
t+11	64.89	97.07	66.75	61.41	71.31	72.28
t+12	68.79	103.78	71.16	67.06	74.71	77.10
t+13	72.84	110.08	75.28	72.72	77.45	81.67
t+14	76.46	115.79	79.33	78.24	80.28	86.02
t+15	80.21	121.28	83.63	83.72	82.76	90.32
t+16	83.92	126.12	87.78	89.04	85.60	94.49
t+17	87.44	130.75	91.83	93.91	87.88	98.36
t+18	91.02	134.69	95.70	98.53	89.57	101.90
t+19	94.71	138.19	99.34	103.19	90.80	105.25
t+20	97.96	141.50	102.85	107.57	92.10	108.40
t+21	101.57	144.67	106.64	111.90	93.36	111.63
t+22	105.32	147.49	110.49	115.79	94.46	114.71
t+23	108.94	150.19	114.07	119.44	95.66	117.66
t+24	112.03	152.72	117.33	123.37	96.84	120.46
t+25	114.67	155.06	120.52	127.16	97.84	123.05
t+26	117.42	157.61	123.47	130.84	99.23	125.71
t+27	120.14	159.87	126.40	134.43	100.40	128.25
t+28	122.25	162.02	128.98	137.75	101.61	130.52
t+29	124.14	164.17	131.17	141.01	102.85	132.67
t+30	125.80	166.45	133.39	144.30	104.08	134.80

D.3 Dense model

Table D.3: MAE on each walk when predicting new hospitalizations for the dense model, for up to 30 horizons. The mean over all walks is also reported. Boldface indicates the best performance on each row. The training dataset covered the European countries, composed of 23 initial regions and 61 augmented regions

Horizon	Dec 2020	Jan 2021	Feb 2021	Mar 2021	Apr 2021	mean
t+1	14.29	14.82	9.28	9.78	17.83	13.20
t+2	18.38	22.67	14.82	13.43	22.43	18.35
t+3	22.61	29.83	19.40	17.21	28.57	23.52
t+4	27.10	38.66	24.96	21.83	34.96	29.50
t+5	32.60	48.27	30.09	26.23	35.94	34.63
t+6	37.56	54.69	35.43	30.43	41.01	39.82
t+7	42.57	62.66	41.11	34.70	47.54	45.71
t+8	45.86	65.79	44.86	37.89	52.78	49.44
t+9	48.91	73.05	48.42	41.08	63.80	55.05
t+10	53.41	78.56	52.28	45.80	68.57	59.72
t+11	57.48	89.02	55.75	50.00	67.13	63.87
t+12	60.60	95.64	59.11	54.47	76.49	69.26
t+13	64.78	96.78	62.81	59.34	80.84	72.91
t+14	66.68	106.79	65.94	63.50	78.68	76.32
t+15	70.66	111.09	69.35	66.04	81.81	79.79
t+16	72.94	115.04	71.70	70.25	85.19	83.02
t+17	76.77	113.80	75.56	74.41	98.18	87.75
t+18	77.98	116.34	77.25	77.74	93.90	88.64
t+19	82.17	117.38	81.32	81.30	105.19	93.47
t+20	83.43	118.56	82.90	86.87	107.68	95.89
t+21	87.01	119.59	84.83	90.05	110.74	98.44
t+22	88.03	120.47	87.44	90.27	114.97	100.24
t+23	89.36	120.98	91.94	95.55	117.80	103.13
t+24	91.41	126.43	91.99	95.56	120.00	105.08
t+25	93.30	126.49	95.55	97.96	121.84	107.03
t+26	93.00	123.07	94.92	103.85	115.77	106.12
t+27	96.45	128.20	96.08	102.77	118.60	108.42
t+28	95.89	124.37	99.25	105.57	129.31	110.88
t+29	100.03	124.79	99.71	107.78	123.62	111.19
t+30	102.54	129.89	97.78	109.30	125.57	113.02

D.4 Encoder-decoder model

Table D.4: MAE on each walk when predicting new hospitalizations for the encoder decoder, using both *Trends* and hospitalizations data, for up to 30 horizons. The mean over all walks is also reported. Boldface indicates the best performance on each row. The training dataset covered the European countries, composed of 23 initial regions and 61 augmented regions

Horizon	Dec 2020	Jan 2021	Feb 2021	Mar 2021	Apr 2021	mean
t+1	71.05	68.37	46.99	50.42	38.89	55.14
t+2	72.40	73.65	46.48	50.14	38.44	56.22
t+3	74.53	76.43	46.11	50.91	39.34	57.47
t+4	78.24	80.22	47.60	52.24	38.43	59.34
t+5	80.28	82.96	49.22	53.66	38.38	60.90
t+6	82.11	85.23	51.21	54.76	38.07	62.28
t+7	83.13	86.74	53.69	55.89	38.25	63.54
t+8	83.67	87.56	56.18	56.05	37.45	64.18
t+9	84.40	88.23	58.56	55.76	36.15	64.62
t+10	85.25	88.90	60.90	55.07	34.79	64.98
t+11	86.06	90.44	63.73	54.01	33.75	65.60
t+12	86.75	91.96	66.68	52.31	32.26	65.99
t+13	87.07	93.11	69.86	50.28	32.08	66.48
t+14	87.49	93.70	73.60	48.21	32.27	67.05
t+15	87.84	94.03	77.39	46.41	32.23	67.58
t+16	88.41	94.03	81.15	45.14	32.52	68.25
t+17	89.36	93.67	84.89	44.43	33.13	69.10
t+18	90.63	92.90	88.65	44.35	33.41	69.99
t+19	91.73	92.12	92.33	44.85	33.36	70.88
t+20	92.58	91.30	95.67	46.66	34.09	72.06
t+21	93.64	90.32	98.69	49.43	34.82	73.38
t+22	94.71	89.13	101.60	52.89	35.78	74.82
t+23	96.16	88.01	104.12	56.54	37.17	76.40
t+24	97.39	86.91	106.38	60.48	38.62	77.96
t+25	98.65	86.24	108.32	64.47	40.22	79.58
t+26	99.20	86.17	110.13	68.39	41.77	81.13
t+27	99.72	86.17	111.69	72.13	43.64	82.67
t+28	99.58	87.29	112.71	76.00	45.21	84.16
t+29	99.81	88.72	113.12	80.23	46.50	85.68
t+30	100.10	90.67	113.29	84.79	47.76	87.32

D.5 Assembled model

Table D.5: MAE on each walk when predicting new hospitalizations for the assembler model, using a baseline model assembled with a dense model, for up to 30 horizons. The mean over all walks is also reported. Boldface indicates the best performance on each row. The training dataset covered the European countries, composed of 23 initial regions and 61 augmented regions

Horizon	Dec 2020	Jan 2021	Feb 2021	Mar 2021	Apr 2021	mean
t+1	12.80	15.05	10.91	9.19	11.98	11.99
t+2	19.70	24.56	17.61	15.06	19.32	19.25
t+3	25.37	33.20	23.62	20.09	25.48	25.55
t+4	31.13	41.90	29.01	25.65	30.75	31.69
t+5	36.71	50.04	34.29	30.47	36.17	37.54
t+6	42.63	57.80	39.39	35.50	41.66	43.40
t+7	47.77	65.43	44.35	40.36	47.45	49.07
t+8	51.93	71.63	48.53	44.71	51.57	53.68
t+9	56.20	78.14	52.64	49.02	55.90	58.38
t+10	60.44	84.04	56.37	53.42	59.92	62.84
t+11	64.17	89.97	59.96	57.77	63.79	67.13
t+12	67.75	95.66	63.09	63.26	66.97	71.35
t+13	71.51	101.35	65.97	68.55	69.69	75.42
t+14	74.67	106.69	68.94	73.75	72.25	79.26
t+15	77.95	111.39	72.21	78.89	74.38	82.97
t+16	81.26	115.48	75.56	83.83	76.62	86.55
t+17	84.60	119.39	79.13	88.29	78.37	89.96
t+18	87.85	122.66	82.70	92.61	80.00	93.16
t+19	91.27	125.37	86.00	97.06	81.11	96.16
t+20	94.38	127.95	89.30	101.30	82.34	99.05
t+21	97.88	130.65	92.87	105.45	83.60	102.09
t+22	101.40	133.13	96.29	109.05	84.97	104.97
t+23	104.87	135.34	99.58	112.39	86.35	107.71
t+24	107.79	137.42	102.80	116.08	87.67	110.35
t+25	110.55	139.50	105.80	119.50	88.33	112.74
t+26	112.72	141.81	108.72	122.92	89.16	115.07
t+27	115.03	143.92	111.44	126.35	91.27	117.60
t+28	116.83	146.10	114.04	129.44	93.37	119.96
t+29	118.66	148.23	115.96	132.48	95.41	122.15
t+30	119.90	150.25	117.95	135.40	97.21	124.14

D.6 Comparison of each model

Table D.6: MAE when predicting new hospitalizations for the different models, for up to 30 horizons. Boldface indicates the best performance on each row. The dataset covered the European countries, composed of 23 initial regions and 61 augmented regions

Horizon	Baseline	Encoder Decoder	LR + Dense	Linear Regression	Dense
t+1	11.75	55.14	11.99	45.89	13.20
t+2	19.21	56.22	19.25	53.14	18.35
t+3	26.05	57.47	25.55	60.42	23.52
t+4	32.65	59.34	31.69	67.84	29.50
t+5	39.05	60.90	37.54	75.33	34.63
t+6	45.47	62.28	43.40	82.90	39.82
t+7	51.66	63.54	49.07	90.37	45.71
t+8	56.95	64.18	53.68	97.83	49.44
t+9	62.21	64.62	58.38	105.53	55.05
t+10	67.37	64.98	62.84	113.28	59.72
t+11	72.28	65.60	67.13	121.10	63.87
t+12	77.10	65.99	71.35	128.99	69.26
t+13	81.67	66.48	75.42	136.83	72.91
t+14	86.02	67.05	79.26	144.73	76.32
t+15	90.32	67.58	82.97	152.56	79.79
t+16	94.49	68.25	86.55	160.53	83.02
t+17	98.36	69.10	89.96	168.47	87.75
t+18	101.90	69.99	93.16	176.51	88.64
t+19	105.25	70.88	96.16	184.61	93.47
t+20	108.40	72.06	99.05	192.77	95.89
t+21	111.63	73.38	102.09	201.12	98.44
t+22	114.71	74.82	104.97	209.54	100.24
t+23	117.66	76.40	107.71	217.90	103.13
t+24	120.46	77.96	110.35	226.23	105.08
t+25	123.05	79.58	112.74	234.44	107.03
t+26	125.71	81.13	115.07	242.63	106.12
t+27	128.25	82.67	117.60	250.62	108.42
t+28	130.52	84.16	119.96	258.36	110.88
t+29	132.67	85.68	122.15	265.91	111.19
t+30	134.80	87.32	124.14	273.54	113.02

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/epl