POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Homework report

Homework 3

Scientific computing tools for advanced mathematical modelling

Authors: Federica Botta, Simone Colombara, Michele Di Sabato

Academic year: 2021-2022

---

1. Mathematical formulation of the problem

The task of this project is twofold:

- **Step 1**: set up an automatic tool that accesses, updates and organizes the COVID-19 epidemiological data of Italy (on a regional basis);
- **Step 2**: predict four variables regarding the COVID19 pandemic situation in three regions in Italy.

In particular, the GitHub repository is updated daily with a new .csv file containing the regional data, such as the number of newly infected individuals, deceased, recovered and so forth. As previously mentioned, we only focused on three regions, which are Lombardia, Lazio and Sicilia, and on four variables:

- new daily infected: number of new positives;
- hospitalized: number of currently hospitalized individuals;
- deceased:the total amount of deceased up to now since the beginning of the pandemic;
- recovered: the total amount of recovered up to now since the beginning of the pandemic;

The assignment is to train a model to forecast the value of these variables on a regional level in seven days, hence we had to minimize the difference between our predictions and the actual values:

$$L_i = \frac{1}{7}\sum_{x=1}^{7} |\hat{F}_i(x) - F_i(x)|^2 \tag{1}$$

where $\hat{F}_i$ and F_i are the predicted and actual values respectively of variable $i = \{\text{new\_daily\_infected, hospitalized, deceased, recovered}\}$.

Given the well-known seasonality of the pandemic variables (see Figure 1), we decided to handle the problem with Deep Learning, due to the capacity of Neural Networks to extract recurrent patterns from the input.

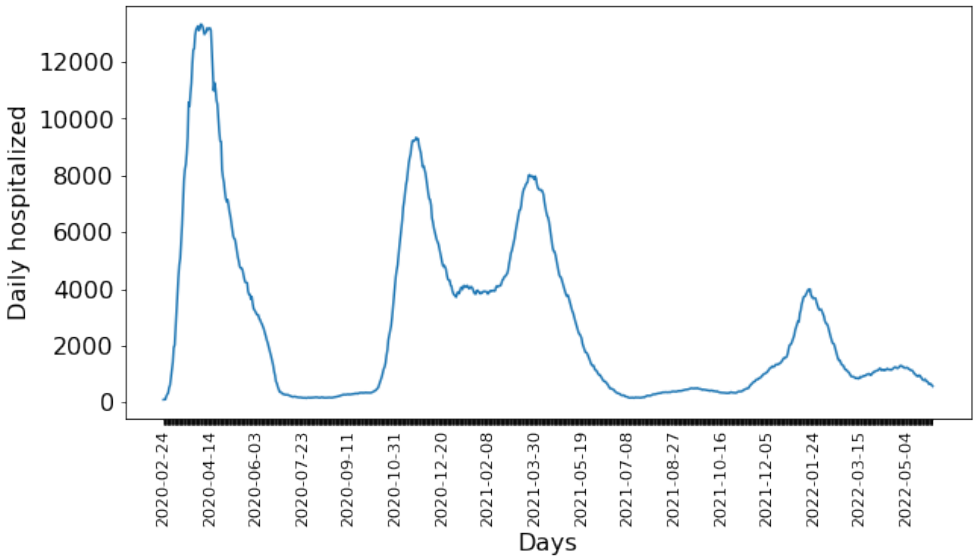


Figure 1: Plot of the hospitalized in Lombardy. There seems to be a clear seasonality.

We now describe the most general mathematical formulation for the assignment:

Let us denote as t_i the i-th day of the pandemic, and as $y_1(t_i)$, $y_2(t_i)$, $y_3(t_i)$ and $y_4(t_i)$ the four variables we are trying to predict. Suppose that $\forall i = 1, ..., N \ \ \exists\ f$ which receives as input the past N days and outputs the 7-days-ahead prediction, such that:

$$f\left(\begin{bmatrix} y_1(t_1,\dots,t_N) \\ y_2(t_1,\dots,t_N) \\ y_3(t_1,\dots,t_N) \\ y_4(t_1,\dots,t_N) \end{bmatrix}\right) \approx \begin{bmatrix} y_1(t_{N+1},\dots,t_{N+7}) \\ y_2(t_{N+1},\dots,t_{N+7}) \\ y_3(t_{N+1},\dots,t_{N+7}) \\ y_4(t_{N+1},\dots,t_{N+7}) \end{bmatrix} = \boldsymbol{y}$$

In conclusion, our aim is to train a suitable neural network (represented by the function f) such that $f \approx y$.

2. Methods

2.1. Step 1

The code we wrote to scrape the GitHub page of the "Protezione Civile" accesses this site [1] and looks for the features we are interested in, which are described in Section 1. For this purpose, we used the python libraries `requests` and `bs4` (in particular the function `bs4.BeautifulSoup`). Since the scraping procedure takes time (roughly 10 minutes to download 800 files, depending on the CPU), we created another function, which checks only for new additions to the GitHub library, comparing the current status of the local folder. If the folder is not up-to-date, the function proceeds with the download of the missing new data. These operations are managed by the functions `scrape.py`, `scraper.py` and `update_files.py`.

2.2. Step 2

To produce the forecast for each feature of each region, we relied on Recurrent Neural Networks (RNN) which have been proven to suffer less from the vanishing gradient problem, in contrast with Feed Forward Neural Networks (FFNN), thus making RNN more suitable for problems such as time series forecasting.

All our models require the same kind of input: we defined as a "sample" a snapshot of the time series, which is constructed by taking a certain number of time stamps (this number is called `window`) and the following `telescope` days (for our purposes, `telescope` $= 7$). These are used in the backpropagation through time to learn the weights. The function `build_sequences` has the role of breaking down the entire time series into such pieces. For example, if the dataset were composed of 1000 time stamps, `window` was equal to 20 and we were trying to predict all the 4 features at once, the input of the network would be a `numpy array` of shape $(50, 20, 4)$

[1] https://github.com/pcm-dpc/COVID-19/tree/master/dati-regioni

for the training and $(50, 7, 4)$ for the target. In this example, we assume that the samples are non-overlapping: this can be generalized by modifying the parameter `stride`, which imposes the number of days in between the starting days of two subsequent samples (in the example above, `stride` = `window`). In our code, the `stride` is equal to 7 days. We experimented with different values of `stride` using multiples of 7, since we noticed a weekly recurrent pattern. The relation among the parameters `window` and `telescope` is crucial, since if we were trying to predict, for example, 7 days using just one week we would expect to achieve worse results than using an entire month.
Furthermore, we noticed that the data coming from the initial weeks of the pandemic were quite underreported. This is reasonable, considering that the tracking protocol of the pandemic was at its initial stages. To avoid using biased or wrong data in the training phase, we decided to discard the initial[2] data contained in the repository. Finally, to avoid problems with the backpropagation of the gradient, we rescaled the data using the min-max approach. Thanks to the rescaling, the first weeks' worth of data are close to zero, hence the model won't learn as much from them. This is an advantage, since the current trend of the features under study seem to be very different when compared to their values during the first months of the pandemic.

We have developed three different approaches to forecast the four compartments:

- **Many to many:** forecast all four features, using all four features. This approach allows us to take into account the correlation among the variables, but requires to have a powerful model. Moreover, to follow this approach one should transform all the features either in daily or in cumulative form, otherwise the network's forecast would be highly influenced by some features and completely miss the others.
- **Many to one:** forecast one single feature at a time, using all four categories. This approach should have the same benefits of the previous option, but should require a less powerful model, as it would predict only one scalar (not a vector).
- **One to one:** forecast one single feature at a time, considering only its past values. With this approach each feature is predicted independently from the others, thus the model is more lightweight than the other two.

For each of these three options, we could train two different types of models:

- One-shot: predict all 7 days at once.
- Autoregressive: use the `window` past days to predict only one, then include this prediction in the training set, roll the training window and predict the second day. Repeat this "rolling prediction" procedure until all 7 days have been predicted.

We implemented all the three methodologies, but we decided to use the one-to-one approach with a one-shot prediction as our final model for the following reasons:

1. the model is trained with less computational cost;
2. the many-to-many and many-to-one approaches require the dataset to be composed of all cumulative features, but since we need to forecast also some daily quantities, at the testing time we need to convert the daily features back to daily values. This step enlarges significantly the errors made on the cumulative predictions;
3. while it is true that the selected features are correlated, the effects of this correlation seem to be evident in time periods longer than one week, so they would likely not influence nor be useful for our prediction (which involves only seven days).

Our model is based on Bidirectional Long Short Term Memory units. As Figure 2 shows, each unit takes as input the output of the previous one and the current input (if `window` = 21, then there are 21 units). Each unit is designed to return its hidden state composed of 128 hidden units: since we used Bidirectional LSTMs, the output of this layer is a collection of 21 vectors, each with $2 \cdot 128 = 256$ components. The next layer is a `LocallyConnected1D` layer with `stride` = 1 and a number of filters equal to 21, which works similarly to a `Conv1D` layer with the same hyperparameters, the only difference is that the weights in a single filter change (hence the different colors of the weight vectors in figure (2)), allowing us to learn a different set of weights for each of the 21 vectors returned by the Bidirectional LSTM units[3]. The output of the `LocallyConnected1D` layer is a single vector with 21 components, which are then connected to the final output layer of the model trough a `Dense` layer with `ReLu` activation function, which returns the 7-days-ahead one-shot prediction. Finally, to avoid overfitting and to tidy the shapes of the tensors in the model architecture, we employed `Dropout` and `Reshape` layers, which are not reported in Figure 2 for the sake of clarity.
The general structure of the model is the same for all regions and all features. Some hyperparameters (such as

[2] we discarded up to three weeks after the first available date, which was the 24th of January 2020.

[3] which should be interpreted as a processed version of the information contained in each day, informed with the values of the previous ones.

the number of hidden units of the LSTMs and the number of neurons in the `Dense` layer) have been changed according to the type of features to treat[4].
The reasoning behind the model we developed is that the LSTM units should be able to elaborate the information contained in each day and the `LocallyConnected1D` layer should learn the best way (i.e. the best weights for the linear combination) to mix the daily information processed and provided by the LSTM units and condense it to one single scalar for each day.

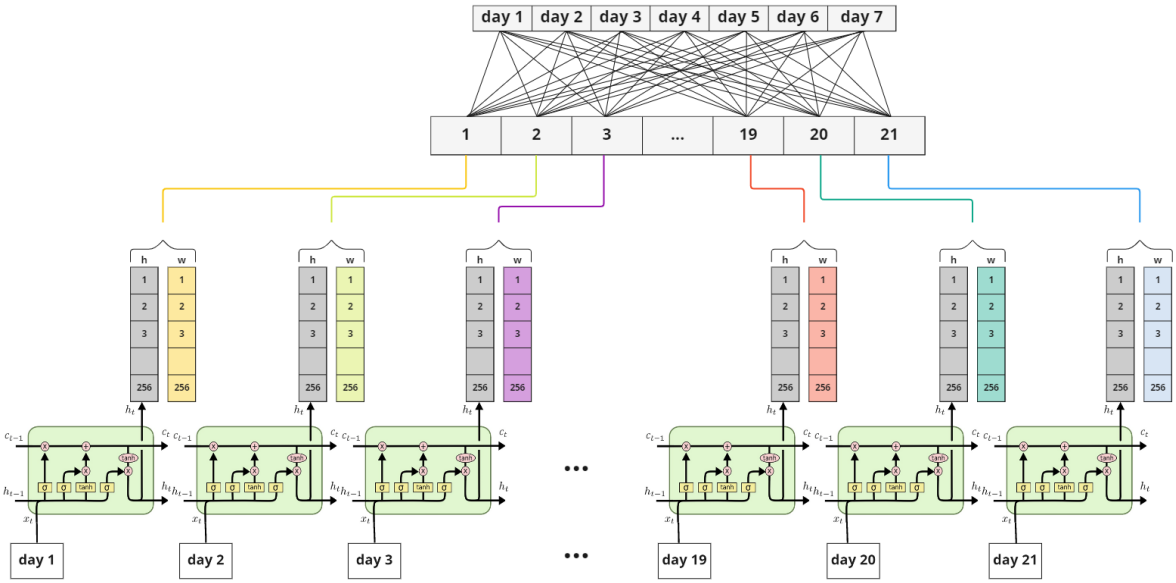
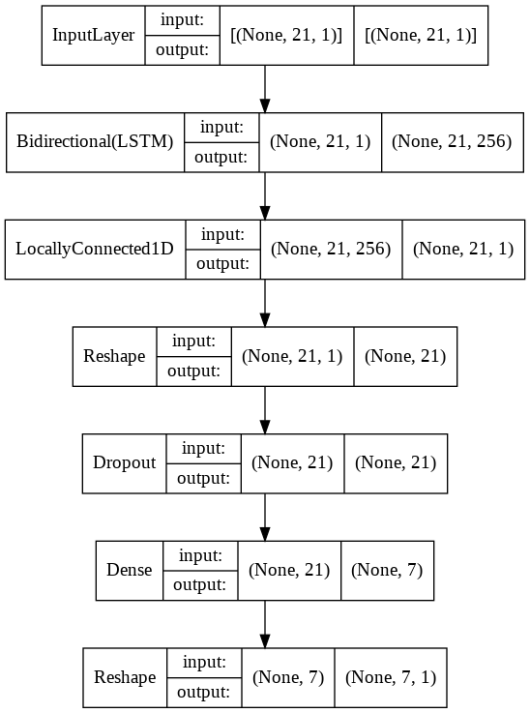


Figure 2: Model architecture with window = 21; **h** stands for hidden state vector and **w** is the set of weights specifically tuned for one single hidden state vector **h**

To train the model, we relied on minimizing the MAE and the RMSE with Early Stopping to avoid overfitting. Additionally, we used an adaptive method to reduce the learning rate when the loss function is on a plateau. Our modeling choices resulted in a learning phase that requires less than one second to loop over the entire dataset.
Finally, Figure 3a and Figure 3b show a more technical plot of the model

[4] if the model is tasked with predicting a daily quantity, then it needs to be more powerful, as it is trying to predict a quantity which varies at a high rate.



(a) Scheme of the RNN used

Layer (type)	Output Shape	Param #
Input (InputLayer)	[(None, 21, 1)]	0
BidirectionalLSTM (Bidirectional)	(None, 21, 256)	133120
LocallyConnected1D (LocallyConnected1D)	(None, 21, 1)	5397
Reshape1 (Reshape)	(None, 21)	0
Dropout (Dropout)	(None, 21)	0
Dense (Dense)	(None, 7)	154
Reshape2 (Reshape)	(None, 7, 1)	0

Total params: 138,671
Trainable params: 138,671
Non-trainable params: 0

(b) Plot of the shaped and number of parameters for each layer

Figure 3: Technical plot of the model

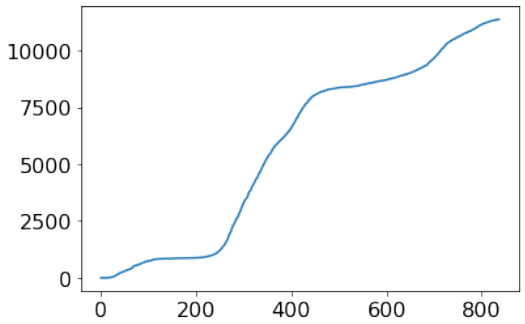
For each of the considered region, we had to set up the model specifically in order to obtain better results.

2.3. Lazio

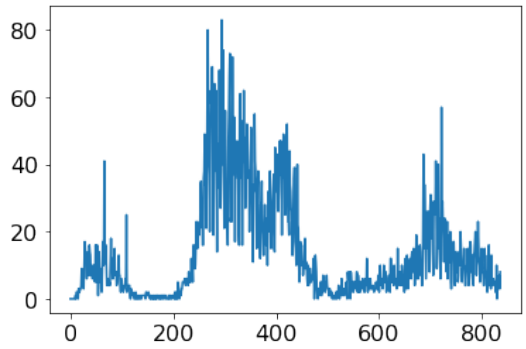
For the Lazio region, we chose to use different strategies, indeed the cumulative datasets (recovered and deceased) were converted in their first difference, in this way our model was learning a more periodic path, such that:

```
dataset_first_difference = np.diff(dataset_cumulative)
```

An example of conversion is:



(a) Original trend of the deceased for Lazio



(b) Modified trend of the deceased into daily values for Lazio

Figure 4: Conversion of cumulative dataset in a daily one

Obviously, in the end we converted the predictions in cumulative and it was obtained an increasing result (cumulative data cannot be decreasing), with:

```
cumulative_pred = np.copy(daily_pred)
for i in range(telescope):
    cumulative_pred[0,i,:] = float(np.sum(daily_pred[0,:i+1,:]) + dataset_cumulative[-1,0])
```

The daily dataset (new infections and hospitalized) are kept invariant.
Finally, an important parameter to tune was the `window`, we trained three different networks for three possible windows = 7, 14, 21 and we kept the one with the best outcome. In general, the 21 window is almost always the best.

2.4. Sicilia

The same model used for the Lazio region in Section 2.3 was used for the Sicilia region.

2.5. Lombardia

For this region we decided to work with the original features: the number of hospitalized and infected individuals is kept as the daily amount, while the number of recovered and deceased is kept cumulative. Moreover, we tried to use an ensemble of models, each with a different `window` parameter: we trained for each feature three models, the first with `window` $= 21$, the second with `window` $= 14$ and the third with `window` $= 7$. The final 7-days-ahead one-shot prediction is an average of each prediction of the three models:

$$\hat{\boldsymbol{y}} = \begin{bmatrix} \hat{y}(t_1) \\ \hat{y}(t_2) \\ \hat{y}(t_3) \\ \hat{y}(t_4) \\ \hat{y}(t_5) \\ \hat{y}(t_6) \\ \hat{y}(t_7) \end{bmatrix} = \begin{bmatrix} \frac{\sum_{m=1}^{M} \hat{y}_m(t_1)}{M} \\ \frac{\sum_{m=1}^{M} \hat{y}_m(t_2)}{M} \\ \frac{\sum_{m=1}^{M} \hat{y}_m(t_3)}{M} \\ \frac{\sum_{m=1}^{M} \hat{y}_m(t_4)}{M} \\ \frac{\sum_{m=1}^{M} \hat{y}_m(t_5)}{M} \\ \frac{\sum_{m=1}^{M} \hat{y}_m(t_6)}{M} \\ \frac{\sum_{m=1}^{M} \hat{y}_m(t_7)}{M} \end{bmatrix}$$

where M is the number of models ($M = 3$), $\hat{y}_m(t_i) \in \mathbb{R}^1$ is the prediction of the value for the selected feature in the day i for $i = 1 : 7$, proposed by model m for $m = 1 : 3$ and $\hat{\boldsymbol{y}}$ is the final prediction of the ensemble. The reason for this approach is that the `window` parameter affects greatly the performance of the model, so instead of only experimenting with different values of this hyperparameter, we decided to exploit various patterns captured by using several windows.

2.6. Options

Our code `prevision_covid.ipynb` gives the possibility to train the model with different kinds of data, depending on the user's preferences. Indeed:

- For a cumulative dataset:
  1. `option = first_difference`: work with the first difference of the feature
  2. `option = None`: work with the dataset as it is
- For a daily dataset:
  1. `option = integrate`: work with the cumulative values of the feature
  2. `option = first_difference`: work with the first difference of the feature
  3. `option = None`: work with the dataset as it is

Also, it can be chosen the region for which the data are used `region = "..."` for all the italian regions.
Finally, it can be chosen to fit a new model or load a pre-existed model, with the `fitting = True/False` parameter.

Remark. *If a model is loaded (* `fitting = False`*), the data given in input must be with the same* `option` *that has been trained.*

In the end of the code, four predictions are proposed:

- `y_hat_21`: for `window = 21`
- `y_hat_14`: for `window = 14`
- `y_hat_7`: for `window = 7`
- `y_hat`: the average prediction such that is equal to `(y_hat_21 + y_hat_14 + y_hat_7) /3`

3. Numerical results

As we expect, the Recursive Neural Network perfectly forecasts the same pattern learned during training, therefore if the pattern changes over time the prediction might become inaccurate. A solution could be to sporadically fine tune the parameters of the network to adjust the predictions. We report some plots which display the ability of our network to understand the weekly pattern even on never-before-seen data.

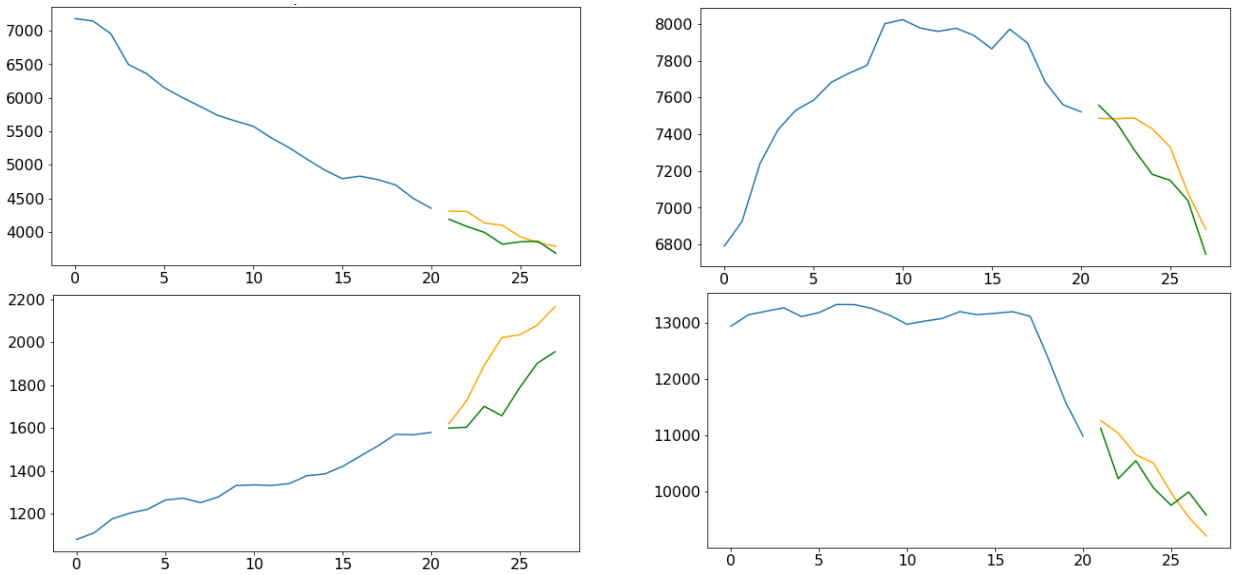


Figure 5: Predictions for the number of hospitalized in Lombardy: the yellow line is the prediction, while the green line is the true value for this feature

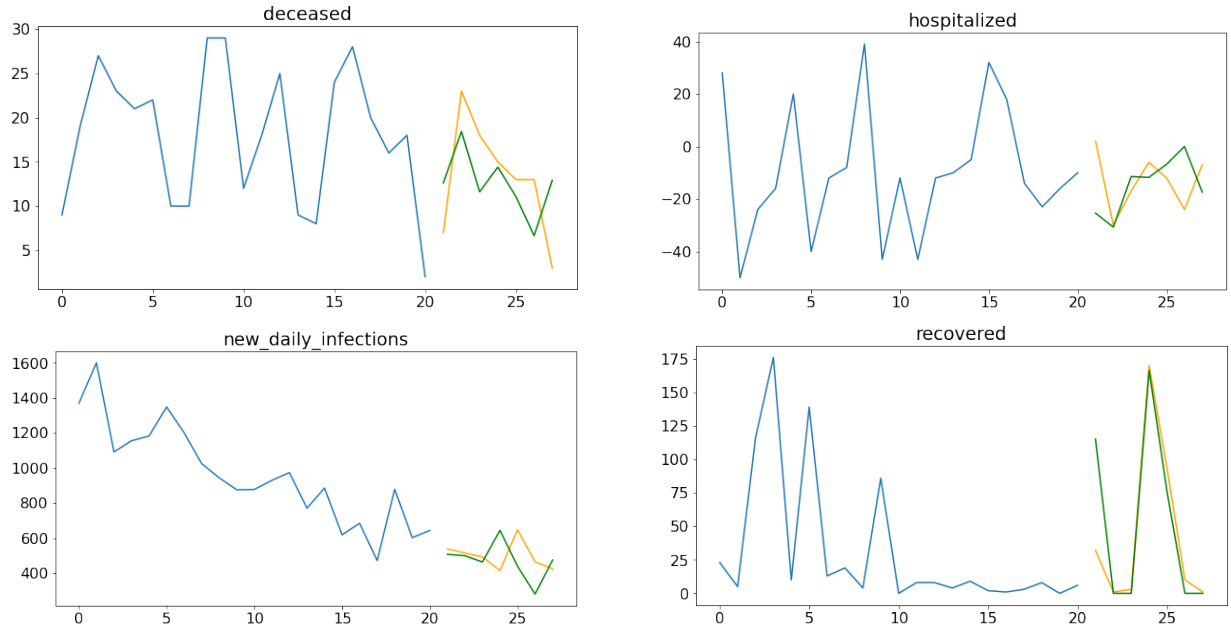


Figure 6: Predictions for the four variables in Sicily

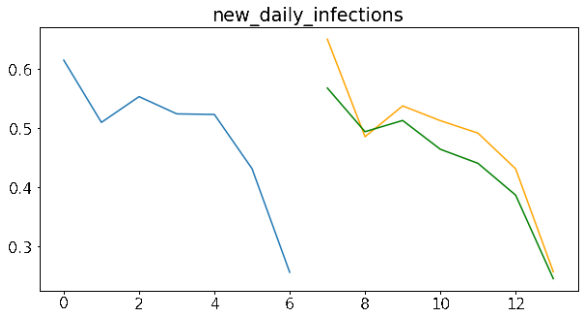


Figure 7: window = 7

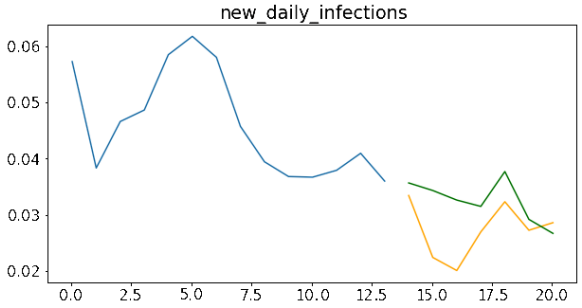


Figure 8: window = 14

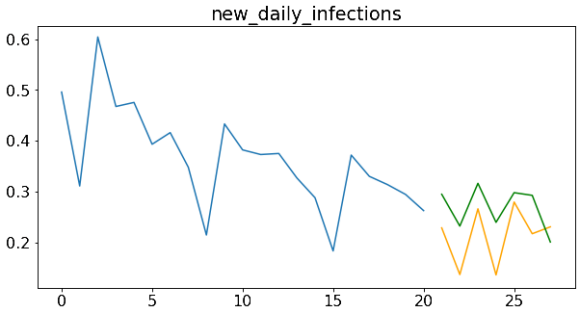


Figure 9: window = 21

Figure 10: Predictions for the new daily infections in Lazio with different windows for three different periods: the yellow line is the prediction, while the green line is the true value for this feature. Note that the values have been standardized.

4. Bibliography

To construct the proposed model, we studied the theory behind LSTM networks using [2] and we read some of their recent applications to the COVID19 epidemic, as in [1].

5. Conclusions

As previously shown in Section 3, our model is able to accurately predict the trend of the time series, especially when the features are transformed into their first difference. This is due to the fact that by differentiating the time series, the network will be able to work on more stationary inputs, thus obtaining better results. This statement has been confirmed by applying the Augmented Dickey Fuller test[5]: the p-value of the dataset composed of the first difference is, among all features, consistently below 0.05.
Moreover, we remark that by adopting a one-to-one approach (as explained in Section 2.2) we achieve results that in terms of accuracy and L2 norm are either the same or better when compared to the many-to-many and many-to-one approaches. The main advantage of the one-to-one approach is the lower computational cost.
Furthermore, since our model is especially good at forecasting the true evolution of each curve, if the trend suffers from a radical change, the pointwise error of our model is expected to grow significantly. The unexpected change in the trend could be due to government policies or even wrongly reported data. For this reason, a possible further development could be to take into account the severity of the restriction imposed by the government and use this as an additional input for the neural network.
Finally, considering the complexity of the task, the type of data involved and the computational cost of training our model (less than one second per epoch), we can say that our results are more than satisfactory, both in terms of RMSE and MAE.

[5] which has as null hypothesis the non-stationarity of the dataset.

References

[1] Nathan Choi. A deep learning model for predicting covid-19 transmission in connecticut. *Honors Scholar Theses*, May 2021.

[2] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. `http://www.deeplearningbook.org`.