Predicting Covid-19 Spread with a Recurrent Neural Network

Harry Heathcock
Department of Computer Science
University of Cape Town
Cape Town, Western Cape, South Africa
HTHHAR001@myuct.ac.za

Brent van der Walt
Department of Computer Science
University of Cape Town
Cape Town, Western Cape, South Africa
VWLBRE001@myuct.ac.za

Kaedon Williams
Department of Computer Science
University of Cape Town
Cape Town, Western Cape, South Africa
WLLKAE001@myuct.ac.za

KEYWORDS

Artificial Intelligence, Recurrent Neural Network, LSTM

1 INTRODUCTION

The whole of 2020 has been controlled by the novel COVID-19 virus pandemic, with many countries being affected both economically and socially. Throughout the pandemic, many countries have shut their borders to others based on how those countries have been dealing with the spread of the virus, but this has been made based on whether the spread has already gone out of control. What if, instead, such decisions as these could have been made based on future predictions? The machine learning problem we have identified and set out to achieve in this project is predicting the average number of total cases a country will have over the next week given past data.

The data-set used to both train, validate and test the data came from the CSSE COVID-19 daily reports from the given Daily Reports data-set. These files were converted from a per day format to a per region format and these were split into the training data (103 regions), the validation data (50 regions) and the test data (50 regions). The training data-set had the most regions as this is used to train the model to fit the trends without over fitting. The model used four values as input: Confirmed Cases, Deaths, Recovered Cases and Active Cases. The same four values were used as output, since each can be predicted, but the focus was on the number of confirmed cases. The evaluation metric used to assess the model was the amount of error, with the prediction being compared to the actual value.

While there aren't many ethical issues that this predictive model can raise, we are aware that there could be issues with trusting the predictions too much, as this could lead to decisions being acted upon the future too harshly or severe precautions taken against people from countries that are predicted explode in infections. While these issues exist as possibilities, they are unlikely, and the benefits of being able to take added precautions would far outweigh these.

2 BASELINE

A simplified linear extrapolation equation was used as the baseline model to compare our trained model's predictions against. The equation is as follows

$$pred(X) = last(X) + \frac{last(X) - first(X)}{len(X) - 1}$$

where X is the set of values to create a next day prediction from, $last(X)$ is the last value of the set, $first(X)$ is the first value of the set, and $len(X)$ is the number of values in the set. This equation calculates the average change over the set of values and then adds that to the last value of the set to create the prediction for the next day. In use, it will be fed in sets of Confirmed Cases, Deaths, Recovered, and Active Cases values, making a prediction for each one.

This model works best when the gradient of the input data changes slowly and approximates a linear function in sections. It shall thus perform best on the Confirmed Cases, Deaths, and Recovered data-sets, as these track ever increasing totals while Active Case values can vary up and down.

3 MODEL DESIGN

The model decided best for this type of sequential problem and the one we used was a Long Short-Term Memory (LSTM) Recurrent Neural Network (RNN). Unlike standard Feed-forward Neural Networks, LSTMs have feedback connections that allow them to process entire sequences of data, which is ideal for making predictions based on time series data. The sequences for our model consisted of a window of 28 days worth of input which were used to make the predictions for the next day. When making a weekly prediction, the first prediction is used to to make a new sequence and this sequence then makes a new prediction for the following and this continues until a week of 7 predicted days is evaluated.

The features for an LSTM are its input, so our model made use of four features, as mentioned previously: Confirmed Cases, Deaths, Recovered, Active Cases . These were represented as an array with 4 values, which led the sequence to be a tensor of the 28 days with each day having 4 values. There were 4 outputs, being the same as the inputs, and these became the 29th day of the sequence, the prediction for the next day.

We identified six hyper-parameters which were tuned to optimize the model as the following: Window/Sequence Size, Learning Rate, Loss Function, Hidden Layer Size, Dropout Rate, and Number of Epochs. The tuning of the hyper parameters and the result of these can be seen in Appendix A.

The window or sequence size is the length of the sequence of input that is used to predict the next day's output. We started with the window size being 7 to match the structure of a week, but after tuning we found that a larger sequence size gave us predicted values closer to the actual values and, more importantly, the standard deviation of these predictions was lower. We believe this to be due to the longer sequence being able to pick up the trends more easily than that of a week period. The window size was tried up to a value of 50, but this performed worse than our optimal found size of 28, and this is likely due to over-fitting occurring.

The learning rate is a hyper-parameter that controls how much the model should change in response to the estimated error each

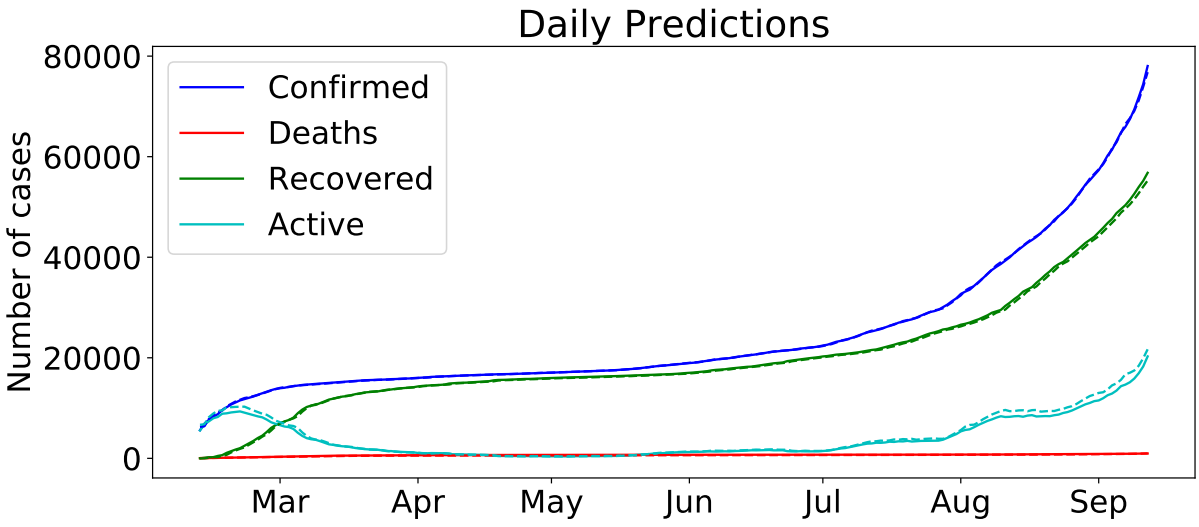

Figure 1: Plot showing the recorded values (solid lines) and predicted values (dashed lines) for Austria. Values are predicted using a 28 day rolling window, predicting only a single day into the future.

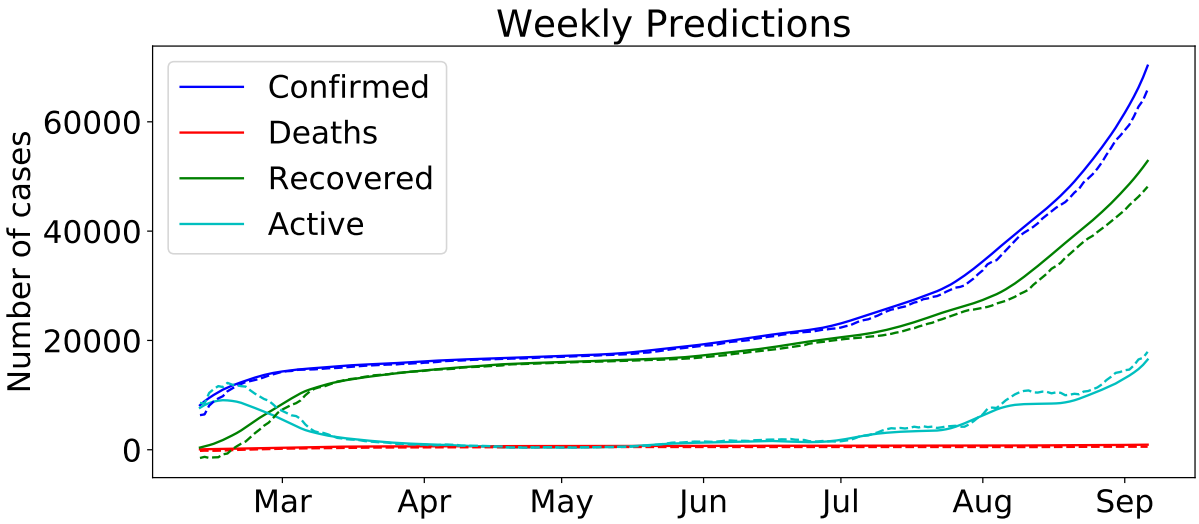

Figure 2: Plot showing the recorded values (solid lines) and predicted values (dashed lines) for Austria. These are generated with a rolling 28 day window predicting a week into the future, with the weekly average on each day graphed for both actual data and predicted data.

time the model weights are updated, essentially this controls how quickly or slowly a model learns a problem. A value too small may result in a long training process that gets stuck, where a value too large may result sub-optimal weights or an unstable training process. We started with a value of 10^{-3} and expected a higher learning rate to produce better results, but instead found that an even smaller learning rate of 10^{-4} had a lower amount of error instead.

The loss function is used to estimate the error for the current state of the model repeatedly so that the weights can be updated. For predicting real-valued quantities, regression loss functions are best used, and of these the two most popular in PyTorch are Mean Square Error Loss (MSELoss) and Huber loss (SmoothL1Loss). MSELoss measures the mean squared error between the prediction and the actual value whereas SmoothL1Loss uses the squared term if the absolute error falls below 1 and an absolute term otherwise, making it less sensitive to outliers than MSELoss. We compared the two loss

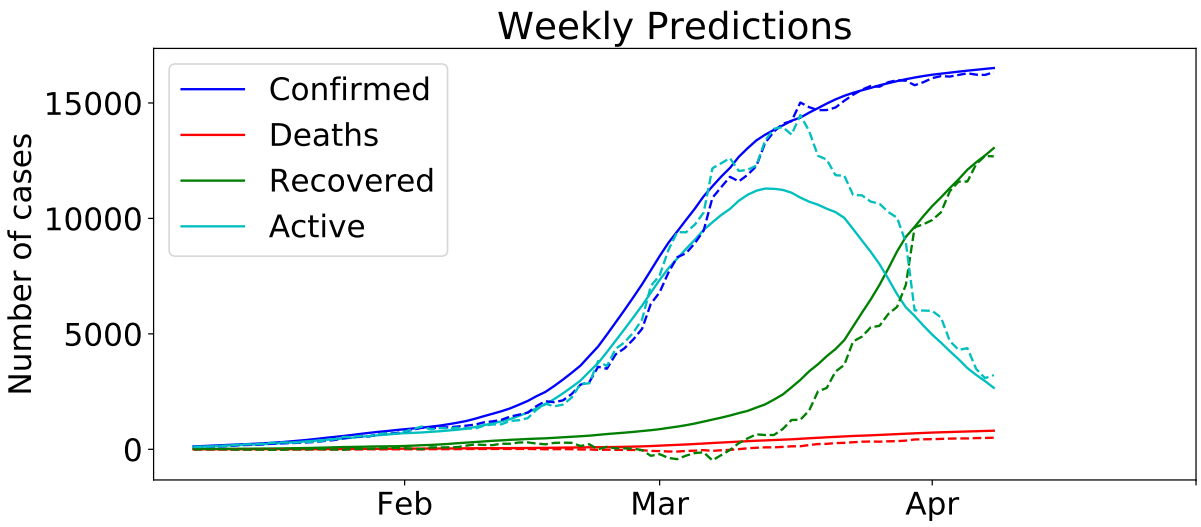

Figure 3: Plot showing the recorded values (solid lines) and predicted values (dashed lines) for Japan. These are generated with a rolling 28 day window predicting a week into the future, with the weekly average on each day graphed for both actual data and predicted data.

functions and found slightly better results from the SmoothL1Loss, likely due to its benefits with outliers, and thus used it as the loss function for our model.

The Hidden Layer Size is the number of hidden units in the model and these units are used to produce output when given the current input. Some people recommend following an equation that you should keep the number of neurons below:

$$N_h = \frac{N_s}{\alpha * (N_i + N_o)}$$

Where N_h is the number of hidden units, N_s is the number of samples in training data-set, N_i is the number of input neurons, N_o is the number of output neurons, and α = is a scaling factor between $2 - 10$.

With $N_s \approx 18000$, $N_i = 4$ and $N_o = 4$, it would be safe to have under 225 hidden units. We started with 125 as a default and testing found that 175 hidden units produced the best result.

Dropout is a regularization method where the proportion of random input and hidden units of a network are set to zero. this has the effect of reducing over-fitting and improving model performance. We found that having dropout gave better results than not having dropout and that a value of 0.35 worked better than other values. This is due to it being optimized high enough to prevent over-fitting and low enough to prevent generalization.

Finally, the number of epochs is a hyper-parameter that controls the number of complete passes through the training data-set. Inversely to dropout, a value too low results in generalization and a value too high results in over-fitting. We made use of early stopping, where after a minimum of 40 epochs, if there was no new minimum loss value model after the epoch for the last 5 passes, the model would stop. This method picks up when the loss of the model starts trending upwards and stops the model in-time before over-fitting can occur, but after testing we found that it produced worse results than using a preset number of epochs decided beforehand based on previous performance. This discrepancy is likely due to the trend moving upwards and the model being saved after this and as such we ended up using 50 epochs for training.

4 EVALUATION AND ANALYSIS

In the evaluation of different hyper-parameters, a focus was given to the consistency of the model (measured with the standard deviation of error) rather than the absolute error itself. This was done as if a model consistently under-predicts by a certain percentage, then the final value can just be increased proportionally to make up for the discrepancy. This was done using the validation data, without looking at how it performed on the test data to ensure that this was not just a correction on the particular dataset.

The final model has error over the full testing data-set as shown below:

Type	Daily Predictions Error	Weekly Predictions Error
Confirmed	$-0.0001\% \pm 0.0466\%$	$-0.0503\% \pm 0.0961\%$
Deaths:	$-0.047\% \pm 0.18\%$	$-0.67\% \pm 2.25\%$
Recovered:	$-0.009\% \pm 0.142\%$	$-0.31\% \pm 1.5\%$
Active:	$0.013\% \pm 0.962\%$	$-0.12\% \pm 1.32\%$

The graphs in Figures 1 and 2 show that the model can quite accurately predict trends in the data before they happen, although tends to underestimate the confirmed and recovered values while overestimating the currently active cases.

Figure 3 seems drastically different, with predicted values being drastically off the actual curves. This is likely due to Japan's response to the pandemic being quite good, causing them to be an outlier in the dataset. This shows that the model should not be

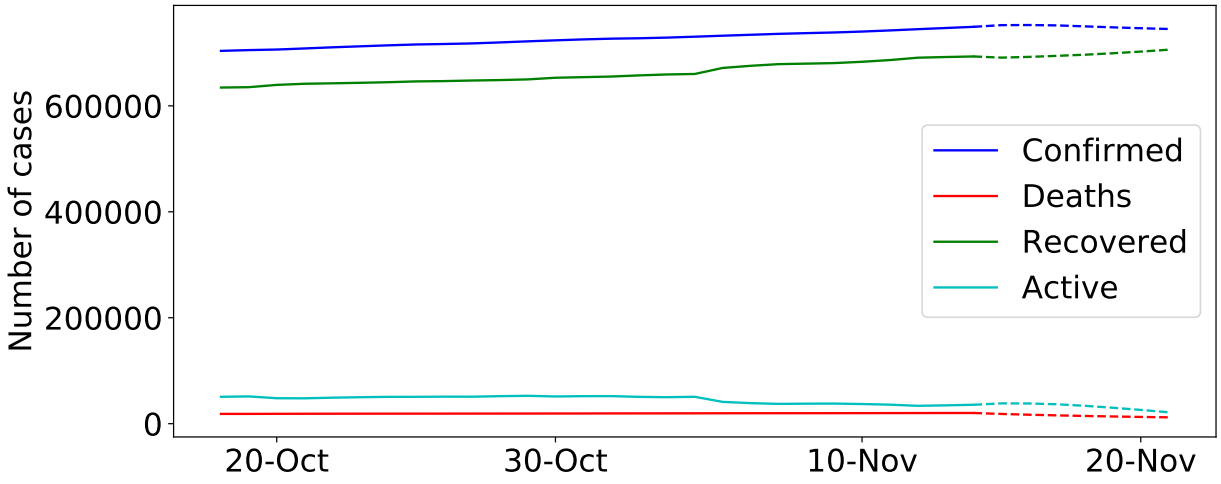

Figure 4: Plot showing the recorded values (solid lines) for the previous 28 days and the model's prediction for the next week (dashed line) for South Africa.

used to guide policies, as it would not be well-suited to predicting changes in this scenario.

The baseline model has error over the full testing data-set as shown below:

Type	Daily Predictions Error	Weekly Predictions Error
Confirmed	$-0.0071\% \pm 0.0445\%$	$-0.027\% \pm 0.102\%$
Deaths:	$-0.0072\% \pm 0.0765\%$	$-0.031\% \pm 0.175\%$
Recovered:	$-0.0106\% \pm 0.0798\%$	$-0.046\% \pm 0.161\%$
Active:	$-0.016\% \pm 0.936\%$	$-0.18\% \pm 2.28\%$

The most interesting takeaway from these results is how similar they are to the trained model and how well such a simple function has been able to perform. The error margins produced over the data-set for each of the four main values is at least comparable to those of the trained model and is in some cases better. This could be due to the small windows of data that we are working with in testing being better approximations of linear functions and it would probably not be as accurate at larger scales, though the usefulness of of increasing the window size would need to be further investigated.

One of the areas in which it does noticeably worse than the trained model is the standard deviation for the Active Cases - this is as expected as the case values fluctuate and do not fit a linear function well. The difference between standard deviation values is not overwhelmingly large, however.

Graphs of the baseline model's predictions can be found in Appendix B.

5 CONCLUSIONS

While the model seems quite accurate at predictions, outliers such as Japan show that it should not be trusted for important decisions as it wouldn't be possible to tell how likely the data-set being looked at is an outlier.

The performance of the model is also shown to be very similar to that of a simple linear extrapolation function. This calls for further investigation into whether there is a use case in this context that the trained model would outperform the baseline model convincingly in.

Another possible improvement to the experiment is to train the model on the changes in the day to day values instead of the values themselves. This would more accurately capture the trends and may limit the negative trend predictions on data which should always increase.

Figure 4 shows the past 28 days of South Africa's data and the model's prediction for the next week. How far this data is off of reality is a good measure of performance as the future data does not yet exist, thus no issues in data could result in accidental use of real information.

A HYPER-PARAMETERS

Hyper-parameter	Values
Window/Sequence Size (WS)	7, **28**, 50
Learning Rate (LR)	0.01, 0.001, **0.0001**
Loss Function (LF)	MSELoss, **SmoothL1Loss**
Hidden Layer Size (HL)	75, 125, **175**
Dropout Rate (Drop)	0.15, 0.25, **0.35**
Number of Epochs	15, 25, **50**, 100

Figure 5: The hyper-parameter values tested, with the chosen values in bold.

Hyper-parameters			Confirmed Cases	Deaths	Removed	Active Cases
LR	HL	Drop				
0.01	125	0.25	−30.41% ± 11.5%	312.39% ± 1761.02%	210.81% ± 1812.13%	1.12% ± 423.4%
0.001	125	0.25	−25.58% ± 3.55%	−22.09% ± 90.25%	−21.6% ± 39.76%	−8.77% ± 84.96%
0.0001	125	0.25	−25.28% ± 3.63%	−16.38% ± 100.3%	−20.99% ± 46.91%	−10.62% ± 73.81%
0.0001	75	0.25	−25.33% ± 3.64%	−19.67% ± 40.62%	−34.36% ± 72.12%	−11.4% ± 64.41%
0.0001	175	0.25	−25.52% ± 3.63%	−0.49% ± 87.71%	−32.2% ± 52.55%	−13.57% ± 57.33%
0.0001	175	0.15	−15.61% ± 4.04%	10.47% ± 115.27%	−16.21% ± 35.66%	1.83% ± 83.15%
0.0001	175	0.35	−35.47% ± 3.13%	−28.13% ± 55.5%	−36.06% ± 28.98%	−20.52% ± 65.63%

Figure 6: The results of a subset of the hyper-parameter tests. ***(WS = 28, LF = SmoothL1Loss, Epochs = 25)***

B BASELINE MODEL GRAPHS

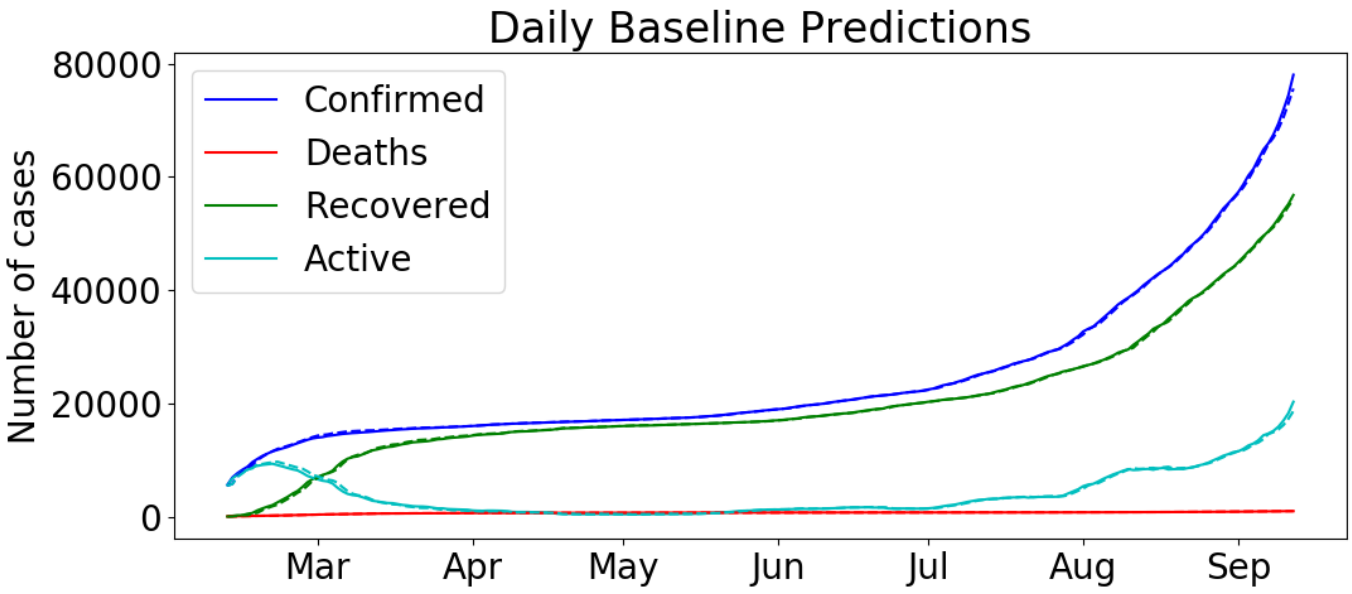

Figure 7: Plot showing the recorded values (solid lines) and predicted values by the baseline model (dashed lines) for Austria.

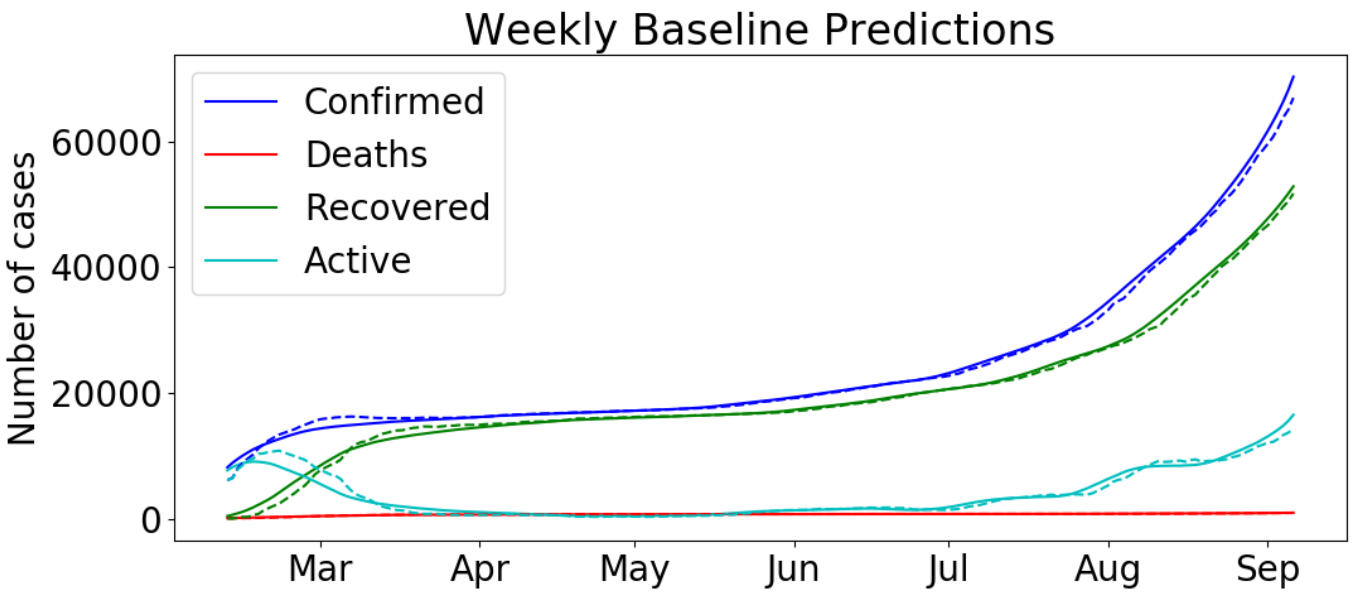

Figure 8: Plot showing the recorded values (solid lines) and predicted values by the baseline model (dashed lines) for Austria.

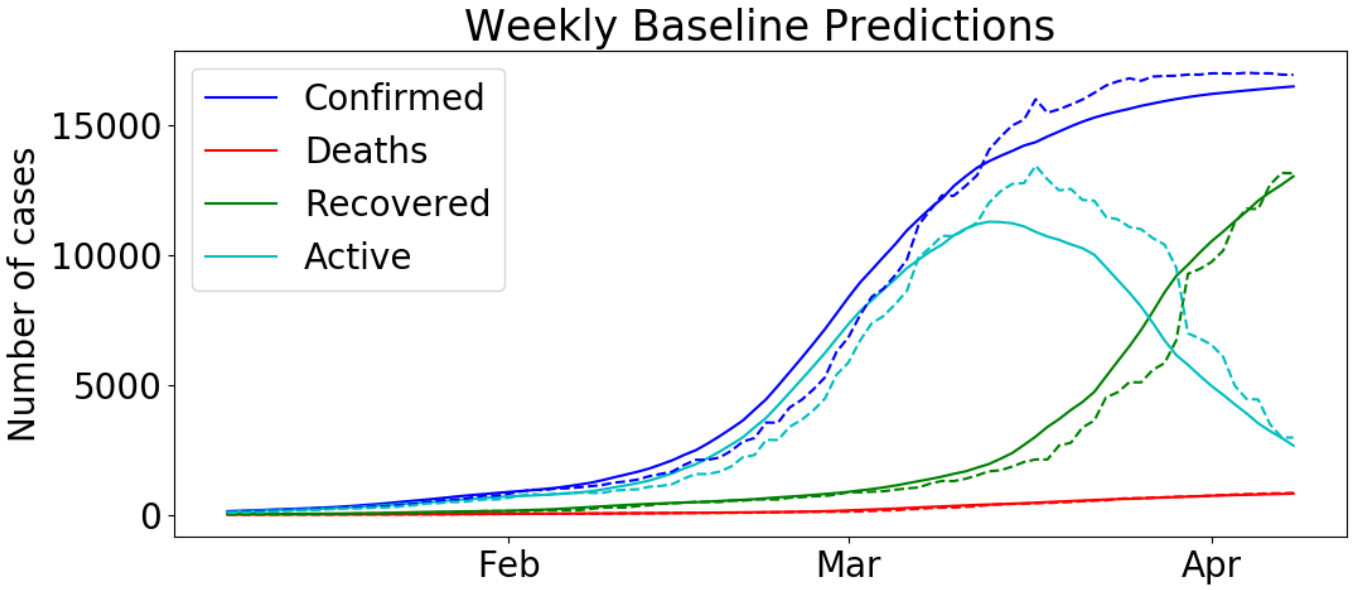

Figure 9: Plot showing the recorded values (solid lines) and predicted values by the baseline model (dashed lines) for Japan.