Forecasting Covid-19 Cases Using Recurrent Neural Networks

Shanchun Yi, Jinghao Zhang

Department of Computer Science
{shanchun.yi, jhgary.zhang}@mail.utoronto.ca

Abstract

It has been proven in many research studies that recurrent neural networks excel at predicting time series data due to the nature of their architectural design. In this project, we conduct a series of experiments with two types of Recurrent Neural Networks, Gated Recurrent Unit(GRU) and Long Short-Term Memory(LSTM), on the Singapore Covid-19 statistics dataset in order to explore the potential of RNNs on forecasting pandemic related data by performing an in-depth analysis of the models' behaviour. Specifically, we compare the performance between models trained with single variable data and with multivariate data, as well as, models outputting different length of future projections. We conclude that an RNN model trained on multivariate data is able to produce more accurate single day predictions, and that large number of training data sequences is required for forecasting longer projections.

1 Introduction

Covid-19 has been raging around the globe for much over a year now. The pandemic has caused unparalleled damage to the global economy. It has took over 141 millions lives worldwide as the writing of this report, without mentioning the prolonged lock downs that have denied most of the social interactions and significantly impact our mental health. Some may wonder if the pandemic is ever going to an end, while many statisticians and health professionals are working on the projected case counts of the pandemic in different areas of the world.
The task itself is a good candidate for machine learning analysis, precisely because the number of factors contributed to the daily cases count are vast; current political status in the region, the lifestyle of its residents, travel frequencies and health care capabilities etc. are all the variables that should be considered in developing an accurate model using the traditional algorithmic and deductive methods. As Canada is facing the incoming third wave, we propose to project reliable predictions via various types of RNN models. This could be surprisingly effective, because by the nature of machine learning, a successful model can provide accurate predictions even without inputs from the corresponding health professionals.

2 Related Works

There are many previous attempts that aimed to predict time series data, specifically data related to Covid-19. An earlier attempt that used LSTM to predict the number of total confirmed cases in Canada, has reported a very low Root Mean Squared Error(RMSE) loss and a high accuracy in its future predictions. Specifically, the paper reported a RMSE loss of 34.83 and an accuracy of 93.4% in short term predictions, a RMSE loss of 45.70 and an accuracy of 92.67% in long term predictions. [1]. However, it is easy to spot limitations in this attempt. The data used was still early on, which means the data presented has a curve with greatly decreased complexity. Secondly, predicting the total confirmed cases is easier than daily cases because the total cases count is a non-decreasing function, which behaves nicely. Rohitash Chandra, Ayush Jain, Divyanshu Singh Chauhan conducted a much more in-depth analysis that applied LSTM, Bidirectional LSTM and Encoder-Decoder LSTM to predict the daily active cases in different areas of India with great level of details [2]. Although the RMSE loss is higher, the models created are working to predict a much more complicated data that spanned eight months. In these eights months, there are times when the number of the cases exploded and when the number of cases plateaued. In this report, we will be conducting research on the data set that is much similar in complexity to theirs.

3 Methods and Algorithms

3.1 Data Definition

The data `covid19_sg.csv` contains the daily cases count, as well as other daily Covid data in Singapore, which could be abstracted as a time-series data. We define the sequence as $\{x_t\}_t$, where t can be ranged from 0 to $N-1$, N is the number of days that have a recorded case count. x_t here is the daily case count at time t.

3.2 Linear Regression

As a baseline for comparison, we first decide to create a single variable linear regression model, which uses daily confirmed cases in the past to predict the daily confirmed cases on a future day. The temporal data is processed and augmented to generate a new dataset based on lagged observation. E.g. time series data [1, 2, 3, 4, 5, 6] becomes X = [[1, 2], [2, 3], [3, 4]], y = [4, 5, 6] with lag variables = 2, and lag time = 2.
We treat the number of lag variables and lag time as hyperparameters. The model is trained with up to 20 lag variables and up to 10 lag time. Rooted mean squared error (RMSE) of the validation set is used as the metric for finding the combination that attains the optimal performance for this model.

3.3 Long Short Term Memory

LSTM is a variant of RNN that aims to solve the exploding/vannishing gradient problem by introducing gates that can keep or forget the previous cell state while updating the current cell state according to the input. In each LSTM cell, the gates operations are listed below,

$$\begin{aligned}
i &= \sigma(W_{ii}x + b_{ii} + W_{hi}h + b_{hi}) \\
f &= \sigma(W_{if}x + b_{if} + W_{hf}h + b_{hf}) \\
g &= tanh(W_{ig}x + b_{ig} + W_{hg}h + b_{hg}) \\
o &= \sigma(W_{io}x + b_{io} + W_{ho}h + b_{ho}) \\
c' &= f * c + i * g \\
h' &= o * tanh(c')
\end{aligned}$$

where σ is sigmoid activation function and $*$ denotes element-wise multiplication, i, f, g, o are the input, forget, cell and output gates. In the specific implementation, each cell's output is also applied with two dense layers with a dropout of $Bernoulli(0.5)$ in between. The dropout layer both acts as a measure to prevent the model overfitting the training data, and as a bootstrapper, whose usage would be elaborated later. We will train the model on 35% of the time series data simply with `lag = 1` and reserve the remaining 65% for testing.

3.4 Gated Recurrent Unit

The goal of the experiments is to train a deep recurrent neural network that is able to take in multivariate time series data and output the predicted future daily confirmed cases. For the GRU model, this is an overview of our approach, under the assumption that number of confirmed cases on a day t is dependent of the data on previous days, $t-1$, $t-2$, ... , $t-k$, for some integer k.

1) Train a model that takes in single variable data (daily confirmed cases only) on multiple past dates to predict that information on the next date.
2) Based on step 1, train a model that takes multivariate data (including daily confirmed cases) on multiple past dates to predict the daily confirmed cases on the next date more accurately.
3) Based on step 1 and step 2, modify the architecture so that it is able to reuse the initial next-day prediction as input to additional RNN units for further future projections on daily confirmed cases.

A special unit structure called Gated Recurrent Unit[3] is used instead of the regular RNN cells to overcome the exploding and vanishing gradient problem. Operations applied at each cell is defined by the following.

$$\begin{aligned}
r_t &= \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr}) \\
z_t &= \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz}) \\
n_t &= tanh(W_{in}x_t + b_{in} + r_t * (W_{hn}h_{(t-1)} + b_{hn})) \\
h_t &= (1 - z_t) * n_t + z_t * h_{(t-1)}
\end{aligned}$$

where h_t is the hidden state at time t, x_t is the input at time t, $h_{(t-1)}$ is the hidden state of the layer at time $(t-1)$, and r_t, z_t, n_t are the reset , update, and new gates, respectively. σ is the sigmoid function and $*$ is the Hadamard product.

The original dataset is preprocessed using window slice augmentation, so that the input length (number of past dates) is fixed and the corresponding label is set to be the next immediate date. This is the same as lagged observation with lag time = 1. The preprocessed training data is split by dates in chronological order and fed into each corresponding GRU unit at the input layer. Therefore, the sequential relation of the input can be captured and learnt by the model.

4 Experiments and Results

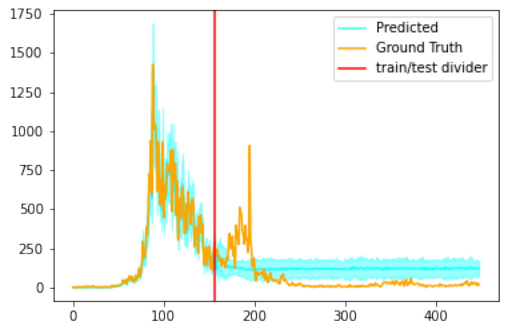
4.1 Linear Regression

The augmented dataset based on lagged observation is split into training set, validation set, and test set without shuffling, as the dataset was transformed from temporal data. In this case, ordering matters and the model should learn from the past experiences to predict future trends.

The model is trained and evaluated with LinearRegression model from Scikit Learn. Models of all possible combinations of hyperparameters in the defined range are run in under 10 seconds. Based on RMSE loss of validation set, the best set of hyperparameters is `lag_variables = 11` and `lag_time = 1`, with validation loss = 12.046. Loss evaluated on the test set is 13.177.

4.2 Long Short Term Memory

To balance between generalization and overfitting, we set a reasonable hidden size. The result can be shown below,



The confidence interval was constructed by applying the model for 100 times. As said before, the existence of the dropout layer has made the prediction random. The confidence interval has values between $(\mu - 2 * \sigma, \mu + 2 * \sigma)$, where μ is the mean, σ is the standard deviation. As we can tell from the graph, the model does much better in recent predictions then predictions far in the future.

4.3 Gated Recurrent Unit

The model is implemented with PyTorch [4].

Similar to the linear model, training set, validation set and test set are split from the augmented dataset without shuffling. In addition, PyTorch Dataset interface and DataLoader class are used to maintain the data batches for training and validation.

The overall architecture of the model is described as follows. It consists of an RNN of `sequence_length` number of GRU as the input layer. Then, some instances of the same RNN stacked on top of the input layer, with outputs from each GRU cell in the previous layer connected one-to-one as inputs to each GRU cell stacked on top. The last cell of the last layer is connected to a dense layer, converting its `hidden_size` (hidden dimension) output to a scalar. Apply ReLU in the end to prevent the output predictions from being negative. Rooted mean squared error (RMSE) is chosen to be the loss function.

Through experiments, we found that a 2-layer RNN component is optimal, as this provides the best performance in terms of training time (computation steps), training outcome, and stability. Training results with the 1-layer RNN component variant does not produce meaningful predictions, which proves that the model has too little expressive capacity. The 3-layer variant takes longer training time and has large fluctuations in validation loss across different runs.

Optimization techniques including random initialization and momentum are applied to the training procedure, in addition to attempts to decrease learning rate. This helped stabilizing loss during training and potentially unstuck the gradients at the ill-conditioned regions of the objective function. More specifically, the Adam[5] optimizer is deployed and the initial hidden states of the GRU cells are set using torch.randn(). Furthermore, the number of training epochs is obtained by closely monitoring the training loss and validation loss curves to apply early stopping in order to prevent overfitting caused by the excess number of training epochs as well as other factors.

The fine-tuned set of hyperparameters are `num_epochs =` 10000, `learning_rate =` 0.0003, `sequence_len = 11`, and `hidden_size = 512`, with validation loss = 8.815. Loss evaluated on the test set is 12.516.

Seeing the success of the 1-feature model, we now want to extend the results even further, by training the model with 2-feature inputs. The second feature is selected to be "number in isolation" as this feature exhibits little dependence on daily cases and is an indication of the number of people with high risk of being exposed to the disease. Training and optimization is done in the same way as the 1-feature model.

Some notable performance of the 2-feature model is achieved with the following hyperparameters: `num_epochs = 25000`, `learning_rate = 0.0001`, `sequence_len = 7`, and `hidden_size = 2048`, with a validation loss of about 8. Due to computational limitations, we are unable to perform as many experiments as previous models. Because of unstable behaviors of the large network during training, we are unable to consolidate on the optimal hyperparameters and unable to reproduce the same result consistently.

5 Conclusion

Although the dropout helps LSTM reducing the effect of overfitting during training, the far future prediction of LSTM tends to be sinusoidal, and sometimes predictions are even squished into a constant value. We speculate that this can be resolved by expanding the dataset. Training based on multiple sequences of time series data may help generalizing the predicted sequence to further future.

With the same number of past dates, the optimized one-feature GRU model is able to achieve better results in comparison to the naive linear model. Further experiments also suggested that the two-feature variant has the potential to provide even more accurate predictions, but requires more sophisticated optimization in the training process as well as more reliable feature extraction.

In conclusion, quality of the predictions made by an optimized RNN model is proportional to the quantity and quality of the training dataset. Higher number of training time series sequences increases the generalization of the RNN model, while higher number of feature dimensions increases the prediction accuracy with appropriate feature selection process.

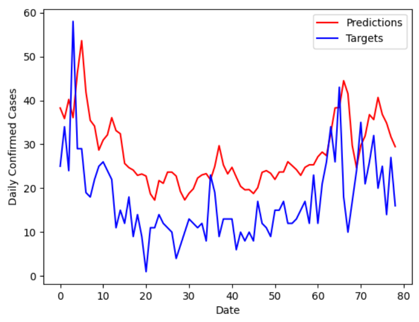


Figure 1: Linear regression test set result

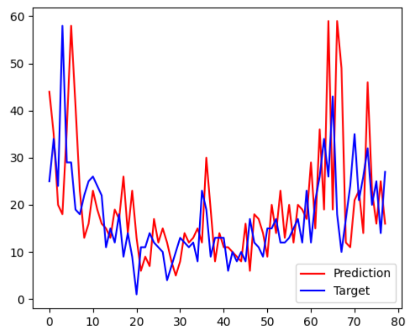


Figure 2: One-feature GRU test set result

6 GitHub Repository

https://github.com/xpbeyo/COVID19-Predictor

References

[1] Vinay Kumar Reddy Chimmula and Lei Zhang. Time series forecasting of covid-19 transmission in canada using lstm networks. *Chaos, Solitons Fractals*, 135:109864, 2020.

[2] Rohitash Chandra, Ayush Jain, and Divyanshu Singh Chauhan. Deep learning via LSTM models for COVID-19 infection forecasting in india. *CoRR*, abs/2101.11881, 2021.

[3] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv:1406.1078 [cs, stat]*, September 2014. arXiv: 1406.1078.

[4] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[5] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, January 2017. arXiv: 1412.6980.