Spatial Analysis of COVID-19 Impact

Sonu Dileep, Anju Gopinath, Poornima Gunhalkar, Karen Rivera
Colorado State University
Term Project – Big Data CS-535

The pandemic ravaging the world emphasizes the need for our governments to have a plan to reduce the severity of the disease. If we divide regions based on population size, we end up with three categories – rural, suburban, and urban. Such spatial analysis of the data allows us to build a more accurate prediction model, thereby allowing governments to be better prepared for future pandemics.

I. INTRODUCTION

The COVID-19 pandemic is the first time in history that we have experienced a problem of this size, and at the same time, we can process and understand the large amount of data gathered. We have various organizations (government, newspapers, academic institutions, etc.) collecting and publishing COVID-19 related data continuously. We need a robust and quick method to process substantial amounts of data and display meaningful and actionable valuable output for the public's health.

Using the Google Open dataset [1], we process infectious and morbidity rates based on population size to analyze the results of 3 different time series models. Counties in the US are identified as rural, suburban, and urban for comparison purposes. We have used three deep learning models – Stacked LSTM, Bidirectional LSTM, and Multivariate CNN to perform comparative analysis. Bidirectional LSTM shows the best performance for rural and suburban regions, whereas Stacked LSTM shows the best performance for urban areas. The accuracy of these models and the speed at which they process the results will guide us in deciding which model works best for the given population size. We also present a visual summary of the results. Using Big Data techniques with distributed models is the straightforward way to process the massive amounts of data. Utilizing a cluster with more than one worker will enable the daily processing of the incoming data quickly. The goal is to stay ahead of the spread of infectious disease in a community.

II. METHODOLOGY

1.Data preprocessing

1.1. Data Cleaning

We only consider counties with 150 days of both valid (>=0) confirmed and deceased cases. We found that Alaska and its small number of counties did not provide sufficient information to be included in the overall processing during data scrubbing. Puerto Rico also had a significant number of "Nan" values in the fields we are using. We removed both territories. The overall number of suburban counties is three times that of rural and urban counties. We are only considering a third of the suburban counties to be consistent with the other categories. After reading the data, we randomized the order of the counties to ensure unbiased reporting.

1.2. Data splitting:

a. Rural, Suburban, Urban data

Since our goal is to perform spatial analysis, we categorize the counties as rural, suburban, and urban. If 0<population<10,000, a county is classified as “rural”, “suburban” if 10,000 <=population <100,000. We classified all other counties as "urban" [2].

b. Training, Validation, and Test sets

We consider only those counties within the US for which we have sufficient data available. So, we have 694 rural counties, 611 suburban counties, and 596 urban counties. From this, we choose 90% for Training and Validation and 10% for testing.

c. Split sequence

Out of 150 days, we use data from 125 days for Training and 25 days for Validation. Let's look at a step size of 10 (in this case). Now, using the set of 150 days, we consider sequence one as days 1 to 10 as input (and predict for next five days 11-15), sequence two as days 2-11 as input (predict for next five days 12 – 16), sequence three as days 3-12 (predict for next five days 17-21) and so on [3].

2. Deep Learning model application

2.1. Inputs and Outputs

The inputs to the models are Total Confirmed cases, Total Deceased, and Population. The outputs (predictions) are Total Confirmed and Total Deceased of the following day.

2.2. Normalization method

Normalization is the process of scaling the range of values to be between 0 to 1. Unscaled data often causes a slow convergence and sometimes prevents efficient learning of the model. We used a custom normalization technique to better train our LSTM models. After splitting the data into multiple small sequences, we scale each input sequence into a range 0 – 1, where the lowest number gets a value of 0, and the highest number gets a value of 1. Also, we divide the output value by the most significant number in the sequence and subtract by one.

2.3 Activation Functions

The activation function transforms the weighted sum of input from a node or multiple nodes to the next cell/layer. The most used activation functions are tanh, Sigmoid, Softmax, ReLU, Swish, etc. Since ReLU helps avoid the vanishing gradient problems, we decided to use ReLU in all the layers of our model.

2.4. Optimization methods

We use optimizers in the neural network model to tune the model weights for minimizing the cost function. They play a significant role in learning, and the parameters of an optimizer decide the model's accuracy and how fast the model converges. So, choosing the right optimizer is a crucial step in hyperparameter tuning. Few commonly used optimization algorithms are Stochastic Gradient Descent (SGD), Nesterov accelerated gradient, Adagrad, RMSProp, and Adam. We used Adam for training our models. Adam is a class of adaptative optimizer which adapts their learning rate while training the model [4].

2.5. Activity Regularization

Regularizes adds a penalty on a layer parameter during optimization. After normalization, the output values fall in the range of 0 to infinity. So, we introduced a custom activity regularization function on the output layers of our model to avoid any negative predictions by the model. A penalty will be added to the final loss function whenever the model predicts a negative value.

3. Summary of Steps - Approach

3.1. Investigation to determine data to be used at the county-aggregation level:

We sifted through the data to find appropriate features at the county aggregation level (Many features had information only at the country or a higher aggregation level, making it unsuitable for in-depth spatial analysis). Also, we reasoned that choosing population as the "other" feature and total confirmed and deceased cases would build better prediction models based on the size of the county.

3.2. Same training and validation data provided to each model:

We ensure that the same data be fed into all three models to perform an "apples-to-apples" comparison.

3.3. Structured code and output file paths to allow for the synchronous running of models:

TensorFlow distributed technology requires that the models running have access to individual log file (.txt files) locations and model checkpoints (.h5 files). So, we made code changes so that the three threads – rural, urban and suburban don't conflict and run simultaneously.

3.4. We trained our model to predict for the next day in sequence for n days:

When making a prediction, we first predict for one day and then include this in the second day's prediction. We retain the result of the prediction of the 2nd day into the prediction for the 3rd and so on. We do this for five days in a sequence.

3.5. Testing the results of the trained model:

After the Training of the model, we process the remaining 10% of counties to calculate the accuracy of the training run. From this, we find MAPE, RMSE, and RAE values that can be compared across models.

4. Deep Learning Models

For predicting the time-series data, recurrent neural networks (RNN) and LSTM perform better in prediction accuracy. However, RNNs cannot handle the vanishing gradients problem, which occurs when dealing with long-sequence data. To solve this problem, LSTM was proposed, which contains the input, output, and forget gate. They help to capture better the correlation of data in the presence of long-term dependencies [5]. Thus, the LSTM models provide better capabilities in remembering the long-term dependencies using memory cells and gates.

The LSTM framework consists of layers of LSTM cells, followed by an activation layer and dropout regularization. We can add layers and hidden units to increase the connections between them and improve the model's accuracy based on the dataset's complexity.

Depending on the individual features of the data, the LSTM layers can be optimized by choosing the number of layers of hidden units, especially for highly complex data.

4.1. Stacked LSTM Model

Stacked LSTM consists of multiple LSTM layers, as shown in Fig. 1a. Combining multiple LSTM layers enables an increase in the model complexity and the depth of the model. Each intermediate LSTM layer outputs resulting sequential vectors used as an input for the next LSTM layer [6].

We divided input data into 90% training set and 10% for testing purposes and used a custom normalization technique to normalize the data. Covid-19 confirmed and death cases of all the US counties, divided into urban, suburban, and rural categories, are given to the model to forecast the future sequences. We designed the Stacked LSTM model using a four-layer stacked LSTM setup with 128, 64, 32, and 32 hidden neurons in the respective layers. The return sequences are set to ‘true’ for the model to ensure that each input data step has LSTM output [6]. Three dense layers are also added to the model to create fully connected networks. The use of the ReLu activation function overcomes the problem of exploding gradient or vanishing gradient problem [7]. Adam is used as an optimizer and mean square error as a loss function to evaluate the model.

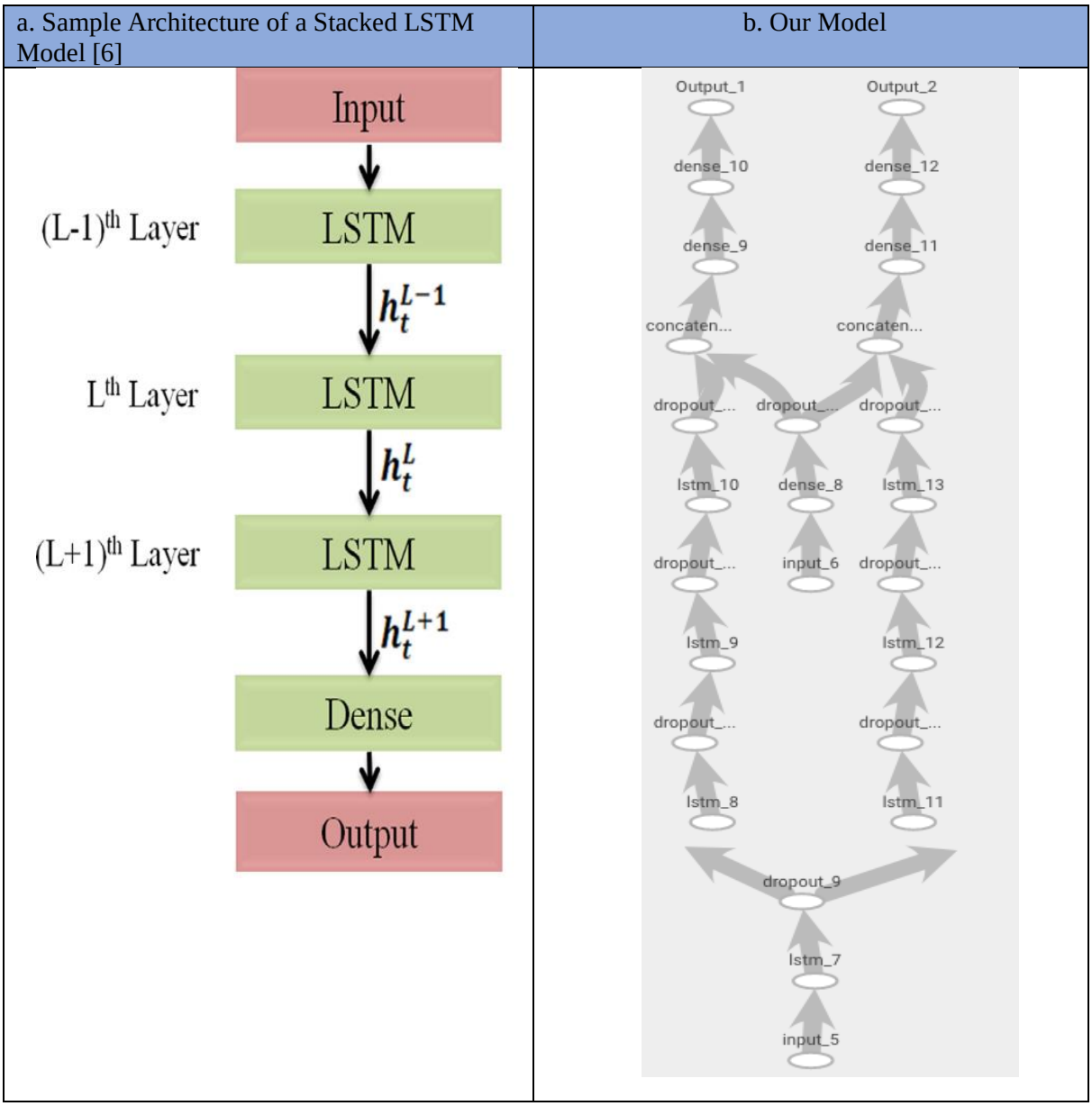


Figure 1. (a) Stacked LSTM. (b) shows the structure of our model implementation, displaying the different layers in the model.

4.2. Bidirectional LSTM Model

Conventional RNNs use only the previous state to determine future states. Whereas Bidirectional RNNs and LSTMs also allow the network to have forward and backward information at each step [6], as shown in Fig 2a.

Bidirectional LSTM networks run the inputs in two ways - one from past to future and another from future to past. Bidirectional LSTM differs from unidirectional because by running information backward, the state

information from the future is preserved. And by using two hidden states combined at any point in time, the network can maintain information from both past and future.

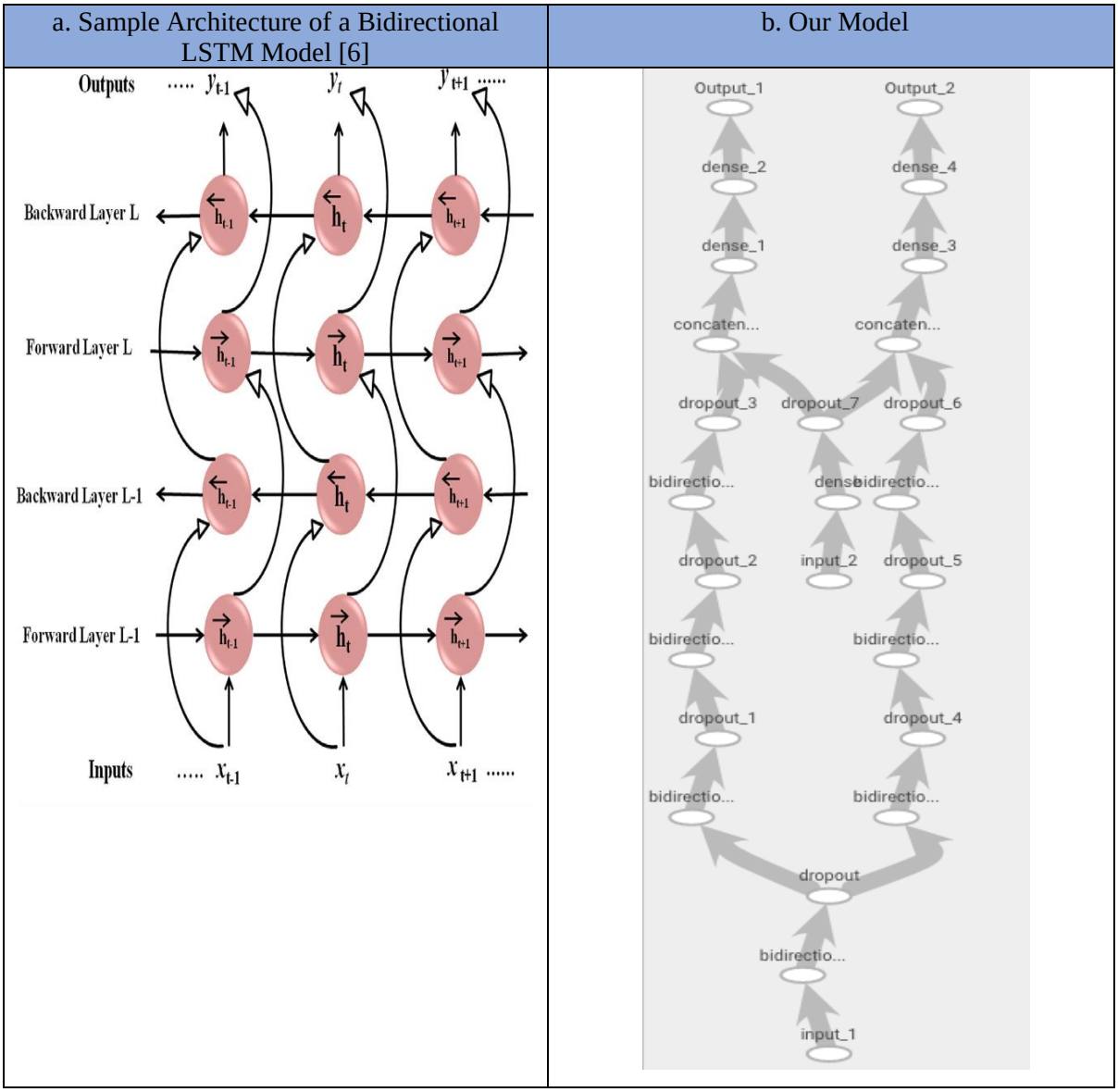


Figure 2. (a) Bidirectional LSTM. (b) shows the structure of our model implementation, displaying the different layers in the model.

4.3. Multivariate CNN Model

Though CNNs are used primarily for image data, many papers have implemented them for time series data because of the property of CNNs, which enable it to learn a function that models the relationship between input features well. It is this success of CNNs that inspired authors to apply it to time-series data [8]. Also, one significant difference from the typical implementation of CNNs is to add the property of multivariate prediction, which can learn the relationship between different variables. This applies to the Covid dataset in that the feature 'population' is added as an input to a new convolutional layer in addition to another convolutional layer that had confirmed and deceased

cases as the input. In addition to 'Conv1D' layers, we add 'Flatten' and 'Dense' layers to transform the data to a 1D vector and for feature interpretation, respectively [9].

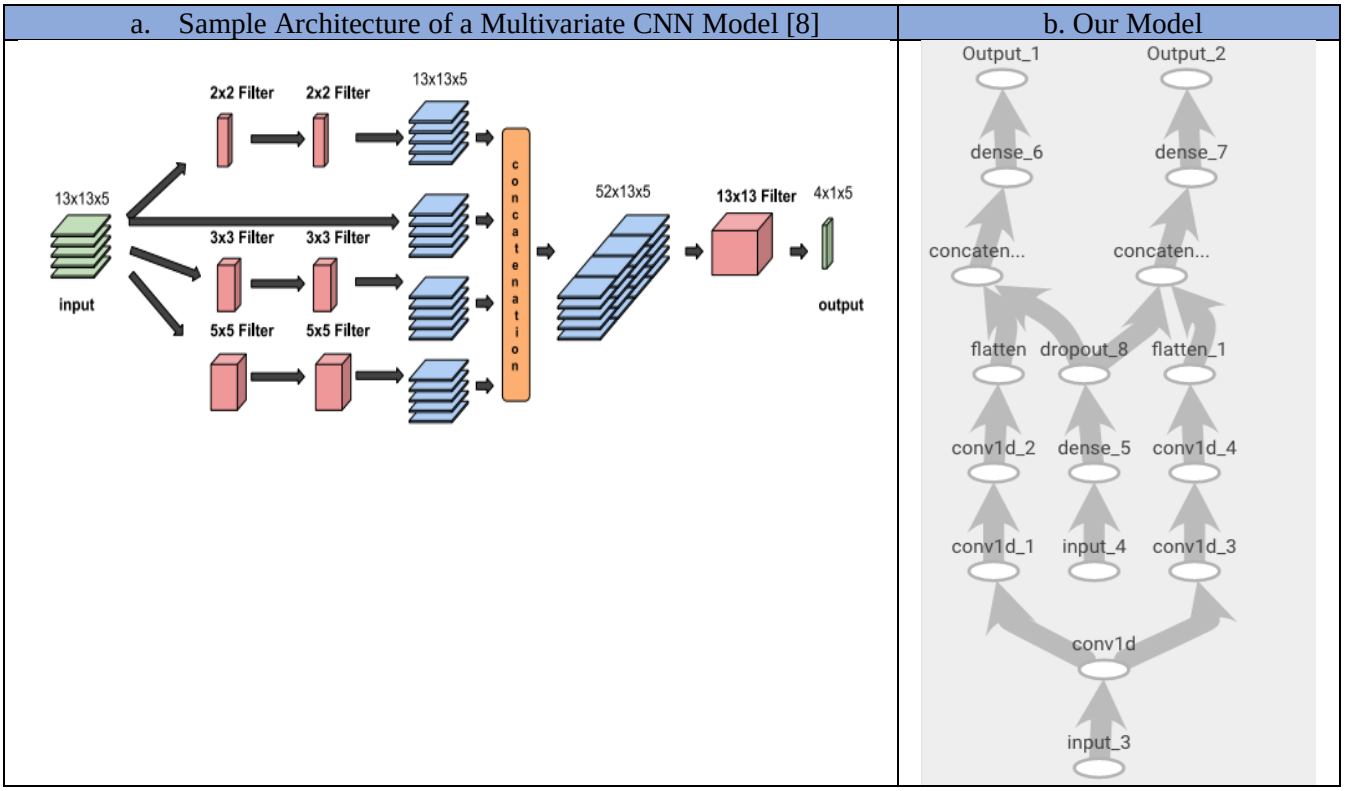


Figure 3.(a) Multivariate CNN. (b) shows the structure of our model implementation, displaying the different layers in the model.

5. Frameworks used in the project

TensorFlow has better community support and documentation compared to other deep learning frameworks. Hence, we used TensorFlow for our distributed implementation.

5.1. Distributed Setup

TensorFlow (Distributed, MirroredWorkerStrategy, GPU)

5.2. Data preprocessing, deep learning models, plots.

Python, Keras, Pandas, Numpy, Sklearn, Matplotlib, Tensorboard

5.3. Development using a small set of data.

Google Colab (CPU and GPU), IntelliJ

6. Dataset used in the project

6.1. Data preprocessing step – we considered the following input data from Google Open dataset

Mobility, Geography, Government Response, and Main csv files.

6.2. Used the following input files from Google Open dataset

Demographics, Index, Epidemiology csv files.

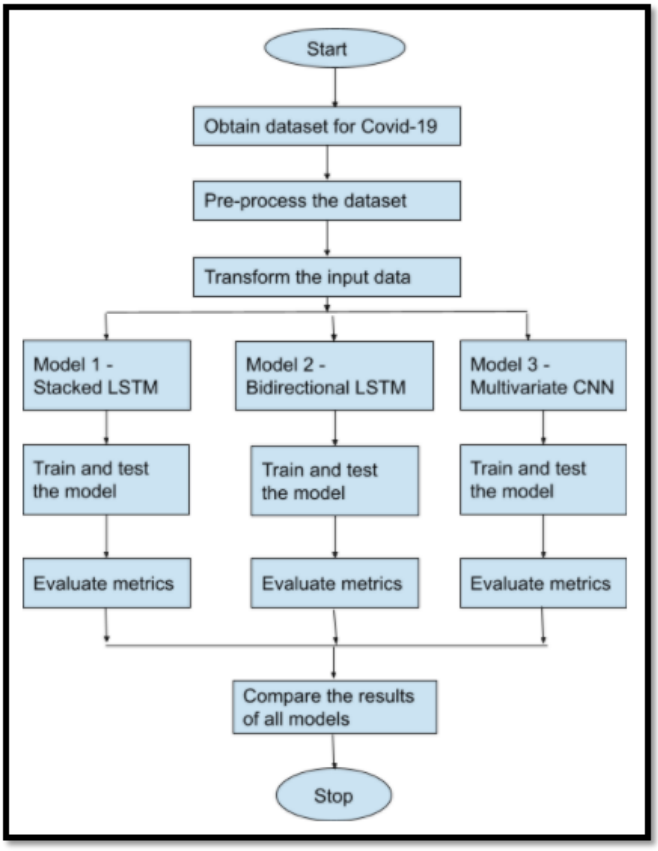
6.3. Structure of the data

Data files are linked using the key (to identify counties) and date fields.

III. EXPERIMENTS, RESULTS, ANALYSIS, AND OBSERVATIONS

We train and test three deep learning models – the two variations of LSTM (Stacked LSTM and Bidirectional LSTM) and multivariate CNN model to see how accurately each can handle the nonlinearity and complexity of COVID-19 time-series data.

We split the dataset into training and validation data from each county for all samples. And finally, for evaluation, we compare the three different evaluation metrics – MAE, MAPE, and RMSE for all three models.



Diagrammatic Approach for the workflow of the proposed methodology

1. Experiments

1.1. Number of epochs

We started with 2500 epochs. We didn't observe a drop in accuracy after reducing the number of epochs to 2000 and further to 1000. We finally settled on this value since it significantly improved the training time without affecting the accuracy.

1.2. Batch Size

The batch size decides how the worker nodes split the work. We experimented with batch size values between 512 to 4096 and found that at 2048, there was a good tradeoff between speed and accuracy.

1.3. Hyperparameters

1.3.1. Number of hidden layers and units in each hidden layer

We started with 1 LSTM layer (Vanilla LSTM) with 64 hidden units. Since the results were not good, we gradually increased the number of layers and also experimented with the number of hidden units in each layer. We found that 4 LSTM layers with the number of hidden units as 128, 64, 32,32 correspondingly, and three dense layers with the number of hidden units as 32, 64, and 32 likewise gave us the best results. We introduced dropouts to avoid overfitting.

1.3.2. Loss function

We attempted Training with different loss functions MSE and MAE. Since MSE provided better results, we used MSE across all models to stay consistent.

1.4. Distributed Setup

We ran the three categories (urban, suburban, rural) simultaneously, each with two workers using TensorFlow's multi worker mirrored strategy [10]. Each category ran the three models with the same data. Overall, Training with this method took about 2 hours per run.

2. Results

The below plots show the loss over epochs as we trained the models using data from 'rural,' 'suburban' and 'urban' counties.

Keys : For Reading Graphs Below

Metric: MSE

Black Graph: Training data

Epoch_Output_1_loss: Confirmed cases

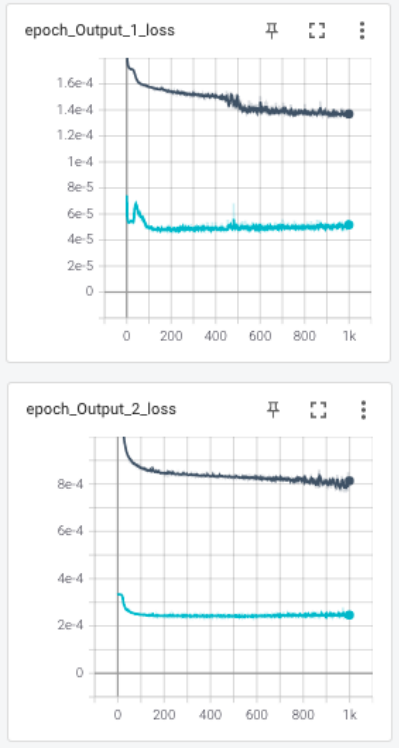
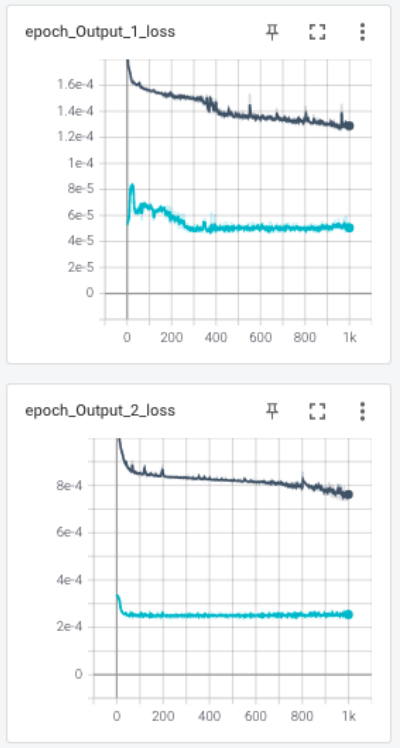
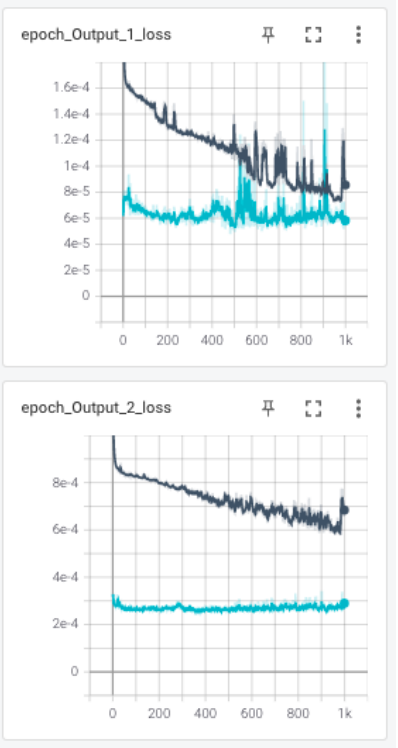
Epoch_Output_2_loss: Deceased cases

Blue Graph: Validation data

Epoch_Output_1_loss: Confirmed cases

Epoch_Output_2_loss: Deceased cases

2.1 Urban

Stacked LSTM	Bidirectional LSTM	Multivariate CNN
		

2.1.1. Model Comparison:

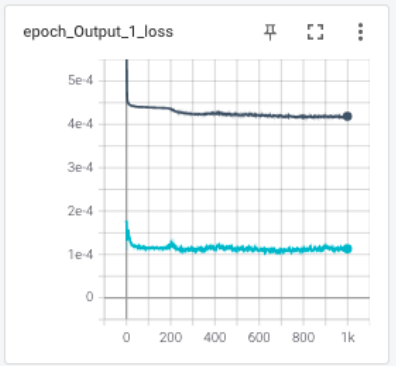
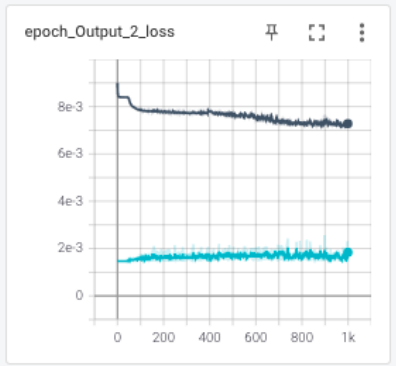
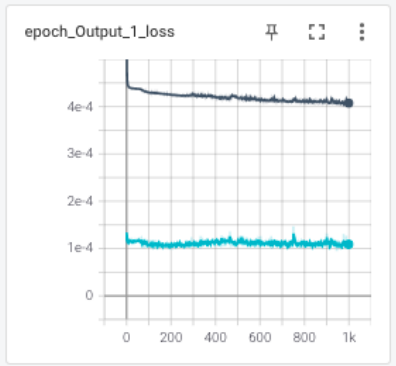
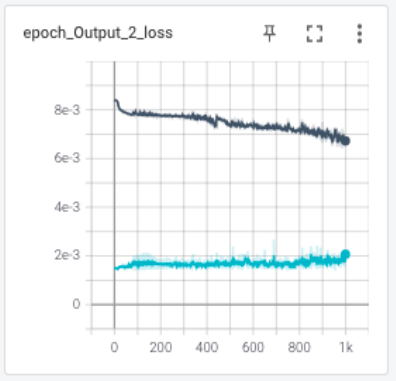
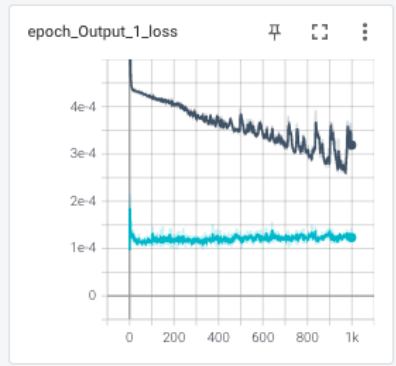
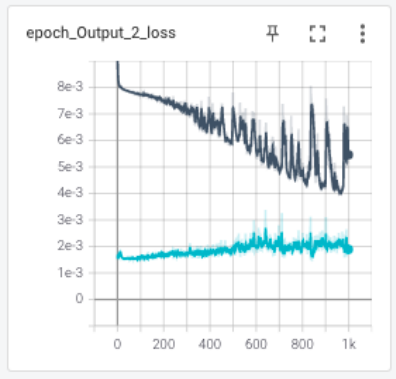
Multivariate CNN is unstable for validation loss for the first output and, minor variations (compared to the other models) are observed for output two.

Stacked LSTM converged faster for confirmed cases (validation loss – output1) as it hit the minimum value in less than 200 epochs.

2.1.2. Comparison Between Outputs (Confirmed vs. Deceased):

The loss for confirmed cases is less than deceased for all three models as the fatality rate is much lower than confirmed cases. This can be explored in future work by using different loss functions. We can try loss functions that consider the relative values rather than the absolute values.

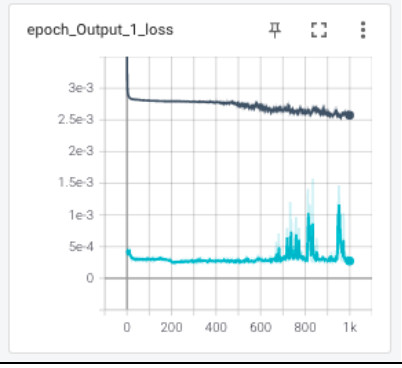
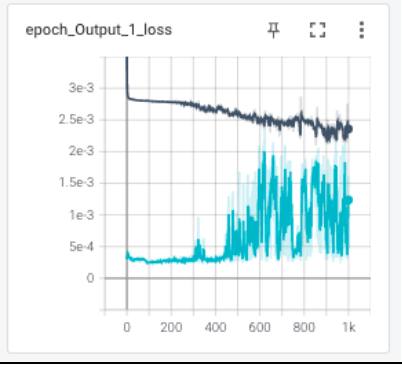
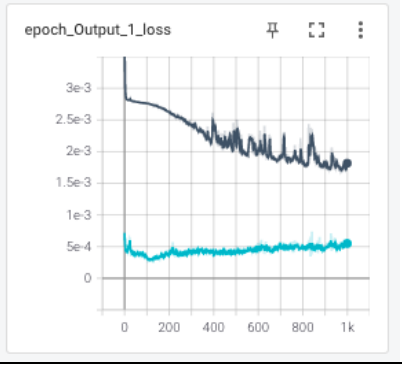
2.2. Suburban

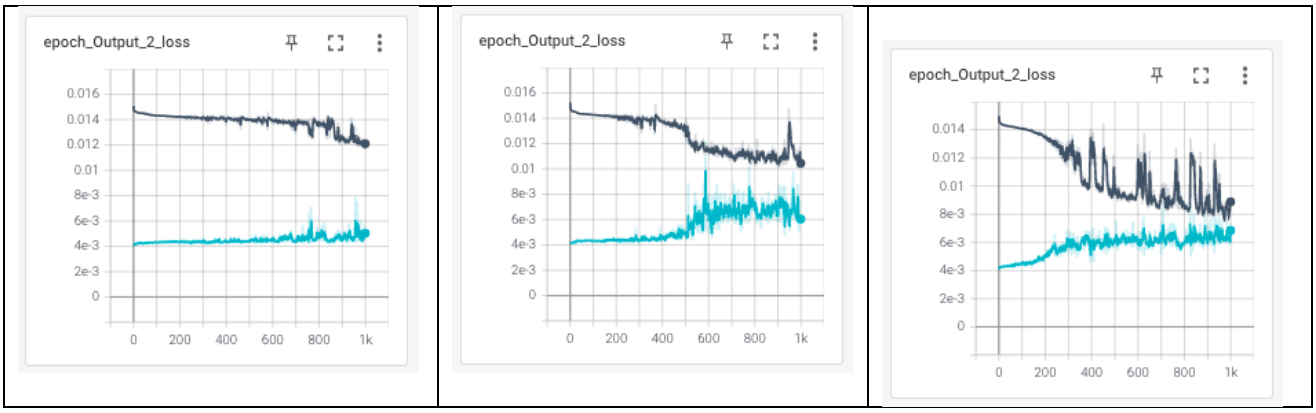
Stacked LSTM	Bidirectional LSTM	Multivariate CNN
 	 	 

2.2.1. Comparison Between Outputs (Confirmed vs. Deceased):

The loss associated with confirmed cases is less than that of deceased for all three models.

2.3. Rural

Stacked LSTM	Bidirectional LSTM	Multivariate CNN
		



2.3.1. Model Comparison:

Bidirectional LSTM is highly unstable for Output 1.

Both Bidirectional LSTM and Multivariate CNN converged at around 100 epochs as compared to 200 for Stacked LSTM.

2.3.2. Comparison Between Outputs (Confirmed vs. Deceased):

The loss associated with confirmed cases is less than that of deceased for all three models.

2.3.3. Epochs:

For output 2, the loss increased with the number of epochs. Further analysis is needed to determine why this happens.

3. Analysis of results

3.1. Evaluation metrics

Loss functions are an evaluation metric that tells us how the model/algorithm performs. Loss functions explain how much the prediction deviates from the actual values. The three loss functions we use to evaluate our models are Root Mean Square Error (RMSE) [11], Mean Absolute error (MAE) [12], and Mean Absolute Percentage Error (MAPE) [13]. RMSE is the average square root of the squared difference between predictions and actual data. MAE is the average sum of absolute differences between predicted data and actual data. MAPE is the mean absolute percentage error between the predicted and actual data.

3.2. Accuracy of Total Confirmed Cases

Variations of LSTM	Rural			Suburban			Urban		
	RMSE	MAE	MAPE	RMSE	MAE	MAPE	RMSE	MAE	MAPE
Stacked	105.51	34.91	0.1378	430.74	139.70	0.1039	**11993.04**	**1780.30**	**0.0920**
Bidirectional	**105.36**	**34.26**	**0.1349**	**430.69**	**139.57**	**0.1038**	12003.04	1795.65	0.0927
Multivariate CNN	105.77	36.07	0.1450	432.00	145.47	0.1098	12038.11	1841.82	0.0942

Table 1. Comparison of Accuracy across Models

3.2.1. Model Comparison:

Bidirectional LSTM performed best for rural and suburban categories, whereas Stacked LSTM performed best for the urban class.

3.2.2. Metric Comparison:

The RMSE and MAE loss values increase from rural to suburban to urban. MAPE shows the reverse trend. MAPE's formula ((predicted-actual)/actual) calculates the average error and is not influenced by the difference of the values.

3.3. Accuracy of Total Deceased Cases

Variations of LSTM	Rural			Suburban			Urban		
	RMSE	MAE	MAPE	RMSE	MAE	MAPE	RMSE	MAE	MAPE
Stacked	**2.54**	**0.62**	nan*	**7.83**	**2.17**	nan*	194.87	25.60	0.0962
Bidirectional	**2.54**	**0.62**	nan*	7.84	2.24	nan*	**194.75**	**25.41**	**0.0960**
Multivariate CNN	2.57	0.66	nan*	7.91	2.51	nan*	194.99	26.59	0.1027

Table 2. Comparison of Accuracy across Models

**nan is because of the calculation of counties with 0 reported deaths. The formula for MAPE has a division by total deaths (=0).*

3.3.1. Model Comparison:

Both Bidirectional LSTM and Stacked LSTM perform best for the rural category. This result emphasizes the fact that these two models' performance is exceptionally close.

Stacked LSTM performs best for the Suburban category, and Bidirectional LSTM performs best for the Urban category.

3.3.2. Metric Comparison:

The RMSE and MAE loss values increase from rural to suburban to urban.

3.3.3. Comparison with values for Confirmed Cases:

Deceased values are less than confirmed values because the deceased rate did not vary as much as the confirmed cases.

3.4. Accuracy of Prediction for five days

Keys – Color Code for Reading Graphs Below

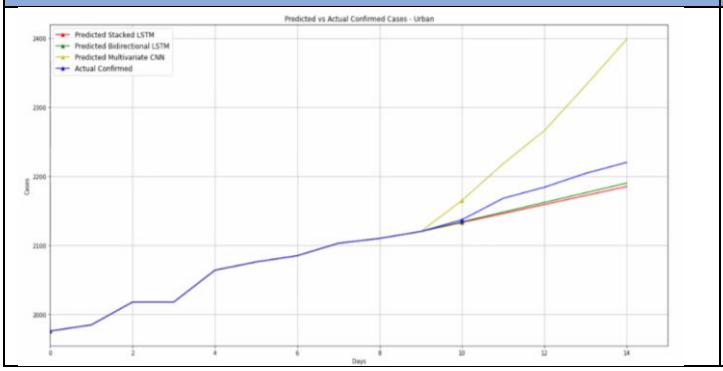
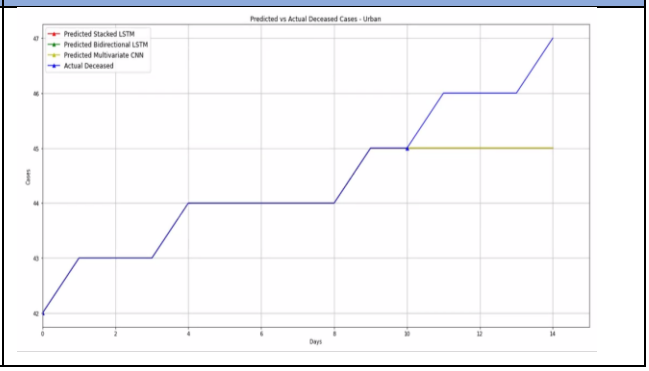
Predicted Stacked LSTM

Predicted Bidirectional LSTM

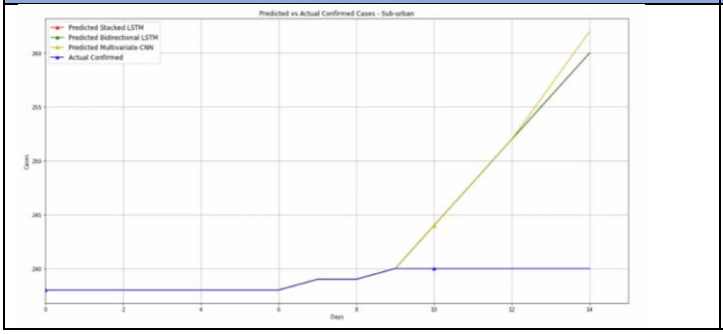
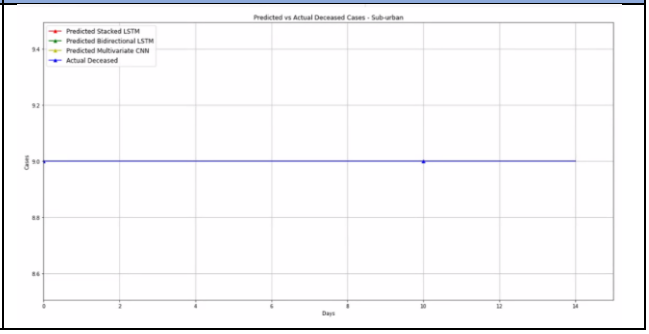
Predicted Multivariate CNN

Actual

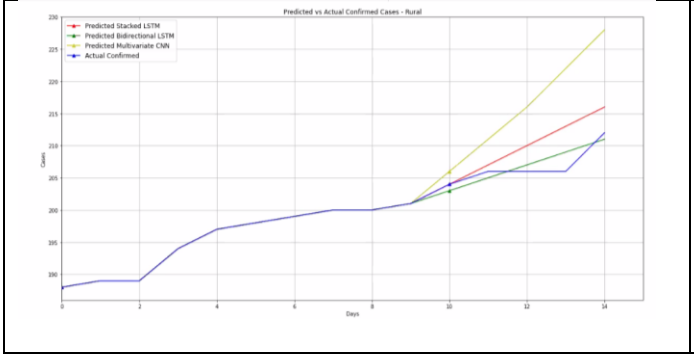
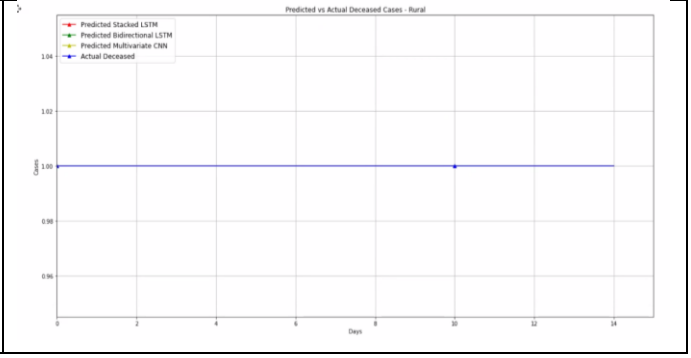
3.4.1. Comparison between 3 models for ‘Urban’ category

Urban Confirmed Cases	Urban Deceased Cases
	

3.4.2. Comparison between 3 models for ‘Suburban’ category

Suburban Confirmed Cases	Suburban Deceased Cases
	

3.4.3. Comparison between 3 models for ‘Rural’ category

Rural Confirmed Cases	Rural Deceased Cases
	

4. Observations

4.1 Why bidirectional takes a longer time to train?

Since the bidirectional model allows the network to pass information in both forward and backward directions, they took longer to train than Stacked LSTM and CNN models. For the same reason, we believe it showed better performance on rural and suburban datasets and comparable performance on the urban dataset. It concatenates and

embeds both interpretations in the hidden states. Since it preserves information from the past, it improves the predictive capabilities [14].

4.2 Why multivariate CNNs perform worse in all categories?

In our experiments training CNNs and LSTMs, we observed that CNNs are not great at time-series predictions compared to LSTMs. LSTMs outperformed CNN for the prediction of confirmed and deceased cases.

4.3 How can we improve the model?

We used the ReLu activation function for each layer. Even though ReLU helps solve the vanishing gradient problem, the dying ReLU problem is a drawback that affects the gradient flow and might affect the model's performance [15]. Using activation functions like Leaky ReLU or Mish activation helps in solving the dying relu problem. Also, training the model with different loss functions or custom loss functions might improve the results further. Experimenting with other models by changing the number of hidden units and dense layers is also a topic for future work.

4.4 Future Work

Several areas can be improved in future work. Our method for distributed Training requires loading the entire dataset into a single node. Sharding the data into individual files would enable the TensorFlow distributed process to use file processing instead of data processing. We would then be able to use a larger dataset – possibly looking at spatial analysis of the counties worldwide instead of just the US. There is also an opportunity to add features specific to social vulnerability categories to the models to see how a particular action (policy change) might influence the results. The Google data provided this information at the State level and higher. We would need to modify our logic to study the trend across countries instead of within a county.

4.5 What are the limitations of LSTM models?

Even though LSTM performed better on time series data compared to CNN, they have a few limitations. LSTMs memorize the patterns in the data sequence from previous days. But when there is a sudden spike in the number of cases or fatality rate due to some external factor, LSTM fails.

IV. CONTRIBUTIONS

Team member	Contribution - Great team effort. Everyone contributed to all of these.
Karen	Data preprocessing and Normalization, Running the training, programming and source code organizing, report writeup, data analysis, distributed framework setup, MAPE and MAE evaluation
Sonu	Stacked LSTM, tuning of the models, report writeup, data analysis, hyperparameter tuning, loss plotting, distributed framework setup, MSE evaluation.
Anju	Multivariate LSTM, report writeup, data analysis, distributed framework setup, MAPE and MAE evaluation, plots
Poornima	TensorFlow vs. PyTorch investigation for models, Bidirectional LSTM, report writeup, data analysis, TensorBoard implementation, distributed framework setup

V. BIBLIOGRAPHY

1. https://github.com/GoogleCloudPlatform/covid-19-open-data
2. https://www.census.gov/programs-surveys/metro-micro.html
3. https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/
4. S. Sun, Z. Cao, H. Zhu, and J. Zhao, "A Survey of Optimization Methods From a Machine Learning Perspective," in *IEEE Transactions on Cybernetics*, vol. 50, no. 8, pp. 3668-3681, Aug. 2020, doi: 10.1109/TCYB.2019.2950779.
5. Hochreiter S, Schmidhuber J. Long short-term memory. Neural Comput 1997;9(8):1735–80. doi: 10.1162/neco.1997.9.8.1735 .
6. Shastri S, Singh K, Kumar S, Kour P, Mansotra V. Time series forecasting of Covid-19 using deep learning models: India-USA comparative case study. *Chaos Solitons Fractals*. 2020;140:110227. doi:10.1016/j.chaos.2020.110227
7. *Hochreiter, S.; Bengio, Y.; Frasconi, P.; Schmidhuber, J. (2001). "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies". In Kremer, S. C.; Kolen, J. F. (eds.). A Field Guide to Dynamical Recurrent Neural Networks. IEEE Press. ISBN 0-7803-5369-2.*
8. C. Liu, W. Hsaio and Y. Tu, "Time Series Classification With Multivariate Convolutional Neural Network," in *IEEE Transactions on Industrial Electronics*, vol. 66, no. 6, pp. 4788-4797, June 2019, doi: 10.1109/TIE.2018.2864702.
9. https://machinelearningmastery.com/how-to-develop-convolutional-neural-network-models-for-time-series-forecasting/
10. https://www.tensorflow.org/guide/distributed_training
11. https://www.statisticshowto.com/probability-and-statistics/regression-analysis/rmse-root-mean-square-error/#:~:text=Root%20Mean%20Square%20Error%20(RMSE)%20is%20the%20standard%20deviation%20of,the%20line%20of%20best%20fit
12. https://www.statisticshowto.com/absolute-error/
13. https://www.statisticshowto.com/mean-absolute-percentage-error-mape/
14. https://towardsdatascience.com/time-series-forecasting-with-deep-stacked-unidirectional-and-bidirectional-lstms-de7c099bd918
15. https://towardsdatascience.com/the-dying-relu-problem-clearly-explained-42d0c54e0d24