A study on Time-Series Neural Networks

Adit Danewa, 2019B5A71378H
Adithya Dev, 2019B4AA0867H
Anirudh Singh, 2019B5A70948H
Bharath Variar, 2019B5A70930H
Kanika Gandhi, 2019B5A71080H
Ojasvi Mittal, 2019A3PS0476H

Project Report
CS F425: Deep Learning
Department of Computer Science and Information Systems
BITS Pilani, Hyderabad Campus
September - December 2022

Contents

1 Predicting COVID-19 cases using Time Series Forecasting

1.1 Introduction

This paper focuses on predicting the spread of COVID-19 using time series forecasting models. Millions of people were infected, and many lost their lives to COVID. Hence identifying trends and predicting future infected cases and the virus spread rate is of utmost importance. An accurate prediction helps optimize the planning of various services and resources to avoid deaths.

1.2 Method

The data set used here is the day-level data on COVID-19 spread for total cases from the whole world and the ten most impacted countries- US, Spain, Italy, France, Germany, Russia, Iran, United Kingdom, Turkey, and India from January 22, 2020, to May 20, 2020. The available day-level data of the adopted ten countries was disaggregated. The initial 5 days of data for each country were discarded because the rate of testing grows slowly initially, and it does not reflect the actual spread. In this study, 80% of the samples are considered for training and 20% for testing. The two models used here were:

1. Facebook Prophet:
    - Facebook Prophet uses several non-linear and linear methods as components with time as a regressor. The model has a number of benefits, including the ability to accommodate multiple period seasonality, custom and known holidays. The model fits reasonably quickly. It also allows flexibility by providing two alternatives for trend:
        (a) A piece-wise linear model
        (b) A saturating growth model
2. Autoregressive Integrated Moving Average (ARIMA):
    - ARIMA(p,d,q) is a combination of Auto-regressive model(AR) and Moving Average model(MA). Here, p is the order of auto-regression, d is the order of differencing, and q is the order of moving average.

ARIMA can be used for prediction only if data is stationary. By observing trends, we know that active, recovered, and death cases are not stationary. Hence one lag differencing is used to convert the data into a stationary form. Later, Dicky-Fuller

test is performed to check the stationarity of the data. PACF and ACF plots have been used to find appropriate values of q and p.
The results for COVID-19 confirmed, active, recovered, and death cases are generated. In order to compare the performance of the models, metrics such as the mean absolute error, root mean square error, root relative squared error, and mean absolute percentage error are used.

1.3 Results

1.3.1 Forecasting active cases

The best MAPE scores are 0.586 for US data by ARIMA and 1.481 for UK data by FBProphet. We can thus infer, with respect to all the errors, ARIMA has far better performance as compared to the FBProphet model.

1.3.2 Forecasting recovered cases

The best MAE results are 78.19 and 69.11 for UK data by ARIMA and FBProphet, respectively. Results show that ARIMA prediction is very close to actual values, with the maximum MAPE value being 15.6 and the minimum value 2.5. Whereas FBProphet did not perform well, the maximum and minimum MAPE for FBProphet are 31.822 and 3.759, respectively. Forecasting of death cases Again, ARIMA gave lower errors as compared to FBProphet.

1.3.3 Forecasting confirmed cases

For this analysis, unlike others, only two countries were chosen, India and the US. FBProphet fits well in the case of the US, whereas ARIMA fits better in the case of India. We can also observe from the result that FBProphet can fit well when the data is less, whereas ARIMA requires sufficient data to model and predict the results.

2 Results on Implementation of paper

2.1 ARIMA

For the implementation of this paper, we used the confirmed cases data for India. Since this paper only uses only 4 months of data for training their model, they obtain forecasts with RMSE values of ≈ 7000.
As our primary improvement, we used data of more than two years for forecasting.

Moreover, we have aslo used square root scaling of the data to further improve the stationarity of the data.
As a result, we obtain RMSE values < 5000 even for non optimal ARIMA models, and a least RMSE value of 4413 for the best model.

2.2 Model Selection

ARIMA(p, d, q) models are defined by three values:

1. Order of Auto-Regression (AR) (p)
2. Order of Differencing (d)
3. Order of Moving-Average (MA) (q)

Differencing is done to make the data stationary. We do this by taking differences of the data, and plot the lags on auto-correlation plots to check stationarity.
From figure 1 we can see that the time series reaches stationarity with two orders of differencing. But on looking at the autocorrelation plot for the second differencing the lag goes into the far negative zone fairly quick, which indicates, the series might have been over differenced.

$$\therefore d = 1$$

To determine `q` we need to use partial auto-correlation plots (PACF plots). Lags are said to be 'significant' when their respective PACF plot is greater than the significance level, shown by the blue region in figure 2.
It can observed that the PACF lag 1 is quite significant since it is well above the significance limit. So we use:

$$\therefore p = 1$$

Finally, to determine q, we use autocorrelation plots, similar to that used to determine d. Figure 3 shows how the lags are plotted, and those above the significance region are counted and found to be 30.

$$\therefore q \in [1, 30]$$

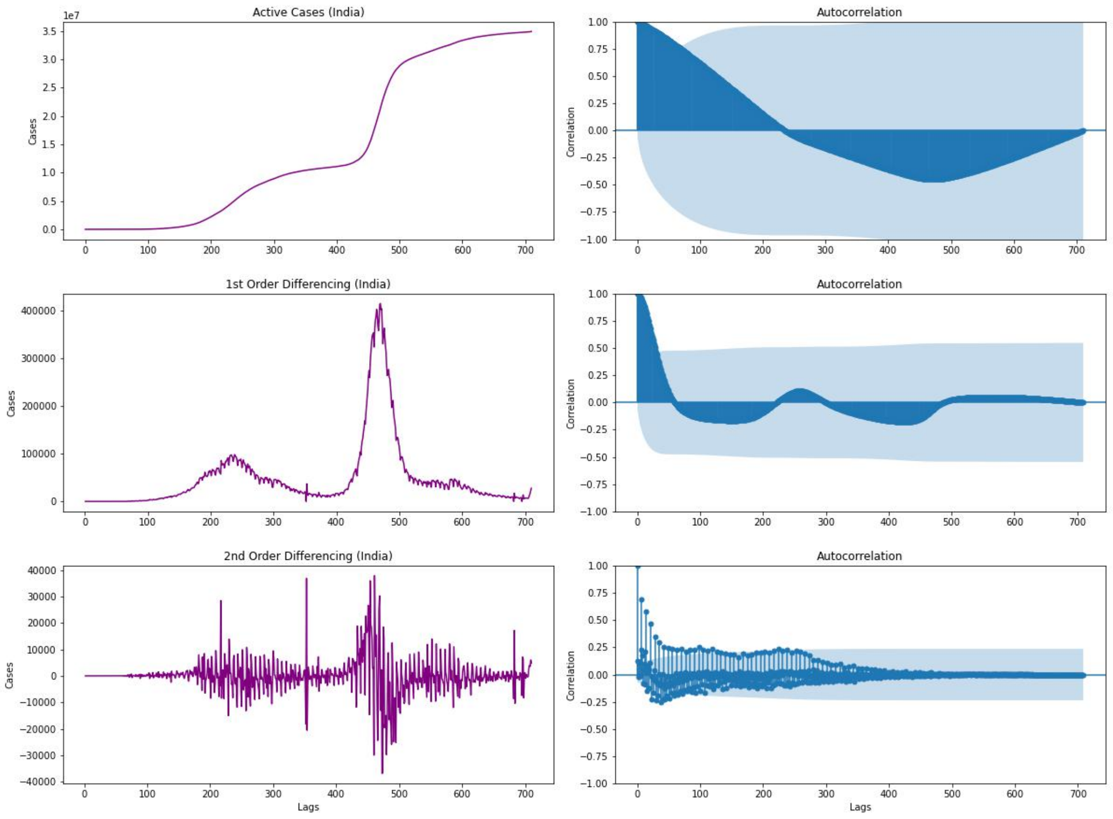


Figure 1: Determining d

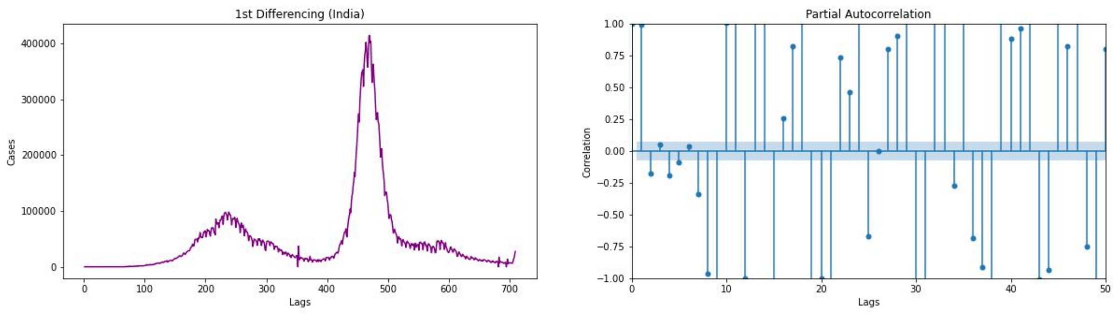


Figure 2: Determining p

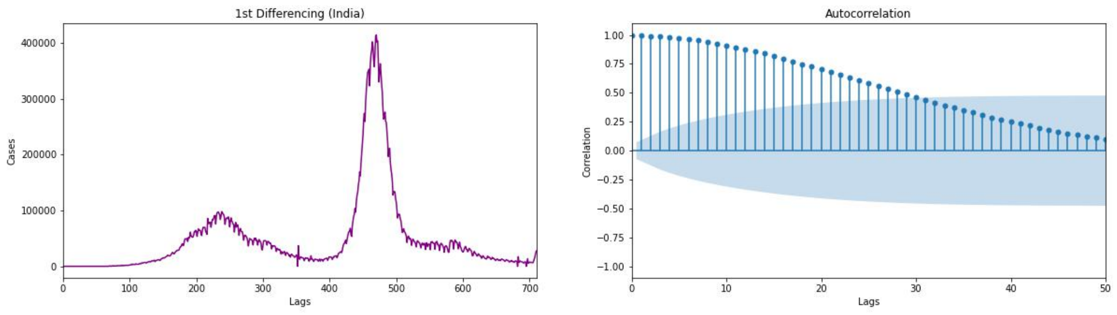
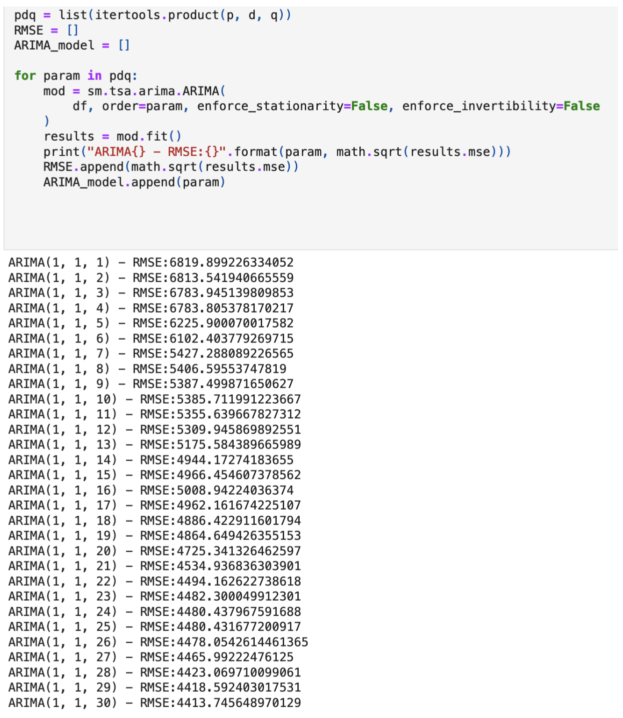


Figure 3: Determining q

We now have have a range of accepted values for `p, d` and `q`. To choose the best model, we train ARIMA models of all possible `(p, d, q)` three-tuples and will choose the one with the least RMSE. We use itertools to iterate through all possible tuples in the specified range and obtain the least RMSE with ARIMA (1, 1, 30), as seen in figure 4.



```
pdq = list(itertools.product(p, d, q))
RMSE = []
ARIMA_model = []

for param in pdq:
    mod = sm.tsa.arima.ARIMA(
        df, order=param, enforce_stationarity=False, enforce_invertibility=False
    )
    results = mod.fit()
    print("ARIMA{} - RMSE:{}".format(param, math.sqrt(results.mse)))
    RMSE.append(math.sqrt(results.mse))
    ARIMA_model.append(param)
```

```
ARIMA(1, 1, 1) - RMSE:6819.899226334052
ARIMA(1, 1, 2) - RMSE:6813.541940665559
ARIMA(1, 1, 3) - RMSE:6783.945139809853
ARIMA(1, 1, 4) - RMSE:6783.805378170217
ARIMA(1, 1, 5) - RMSE:6225.900070017582
ARIMA(1, 1, 6) - RMSE:6102.403779269715
ARIMA(1, 1, 7) - RMSE:5427.288089226565
ARIMA(1, 1, 8) - RMSE:5406.59553747819
ARIMA(1, 1, 9) - RMSE:5387.499871650627
ARIMA(1, 1, 10) - RMSE:5385.711991223667
ARIMA(1, 1, 11) - RMSE:5355.639667827312
ARIMA(1, 1, 12) - RMSE:5309.945869892551
ARIMA(1, 1, 13) - RMSE:5175.584389665989
ARIMA(1, 1, 14) - RMSE:4944.17274183655
ARIMA(1, 1, 15) - RMSE:4966.454607378562
ARIMA(1, 1, 16) - RMSE:5008.94224036374
ARIMA(1, 1, 17) - RMSE:4962.161674225107
ARIMA(1, 1, 18) - RMSE:4886.422911601794
ARIMA(1, 1, 19) - RMSE:4864.649426355153
ARIMA(1, 1, 20) - RMSE:4725.341326462597
ARIMA(1, 1, 21) - RMSE:4534.936836303901
ARIMA(1, 1, 22) - RMSE:4494.162622738618
ARIMA(1, 1, 23) - RMSE:4482.300049912301
ARIMA(1, 1, 24) - RMSE:4480.437967591688
ARIMA(1, 1, 25) - RMSE:4480.431677200917
ARIMA(1, 1, 26) - RMSE:4478.0542614461365
ARIMA(1, 1, 27) - RMSE:4465.99222476125
ARIMA(1, 1, 28) - RMSE:4423.069710099061
ARIMA(1, 1, 29) - RMSE:4418.592403017531
ARIMA(1, 1, 30) - RMSE:4413.745648970129
```

Figure 4: Model Selection

2.3 Fitting model

We now fit the model using the function `model.fit()`. Figure 5 shows the results of the ARIMA Diagnostics.

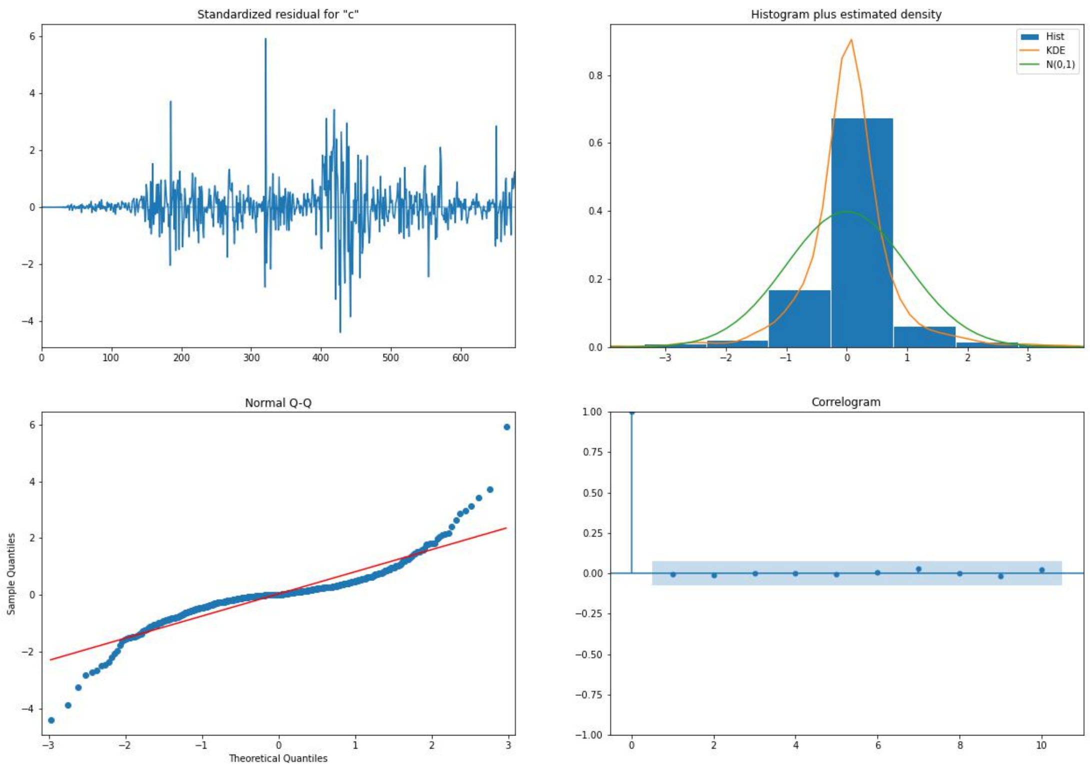


Figure 5: ARIMA diagnostics

2.4 Results

Finally we use the model to predict confirmed cases and forecast for the future. The results can be seen in figures 6 and 7.

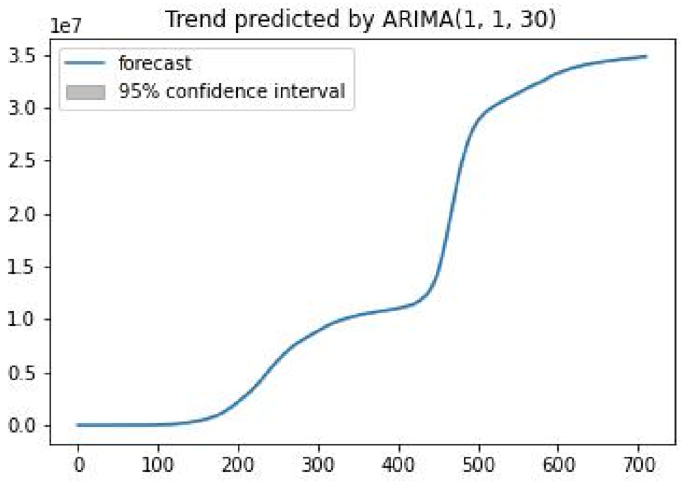


Figure 6: Predicted confirmed cases in India

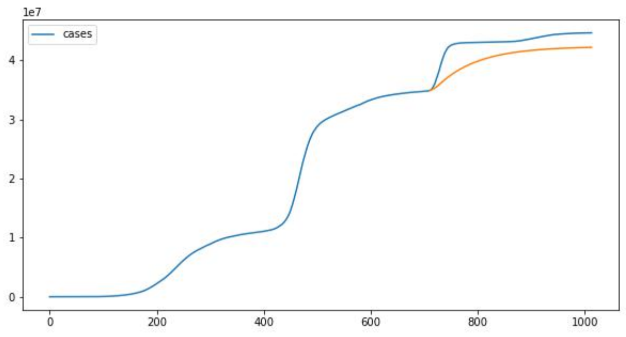


Figure 7: Forecasted Cases for the future in India

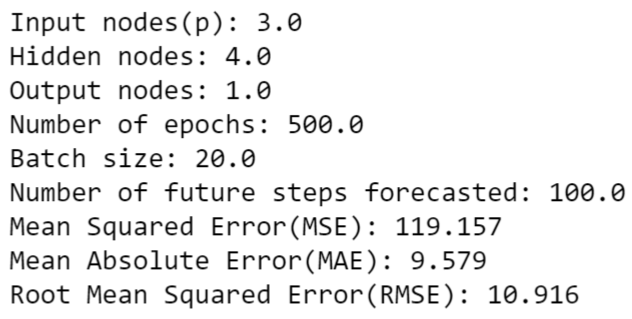
3 ANN

ANNs are another alternative used for time series forecasting. They are good at capturing the non linearity of time series data. We are using a feed-forward neural network as our model.
The output layer consists of 1 unit and input layer consists of p units where p is the number of previous time steps used. For each date value, the model takes the number of cases for all the past days as input and uses corresponding number of cases values

as output labels. We try different configurations of the hidden layers and select the one with the lowest RMSE values.

3.1 Results

Our neural networks perform remarkably well with remarkably low RMSE values of just 10.916. However, these models were very slow, taking upto 2 hours to run on our systems whereas the ARIMA Models ran in less than an hour.



```
Input nodes(p): 3.0
Hidden nodes: 4.0
Output nodes: 1.0
Number of epochs: 500.0
Batch size: 20.0
Number of future steps forecasted: 100.0
Mean Squared Error(MSE): 119.157
Mean Absolute Error(MAE): 9.579
Root Mean Squared Error(RMSE): 10.916
```

Figure 8: Forecast Results by ANN

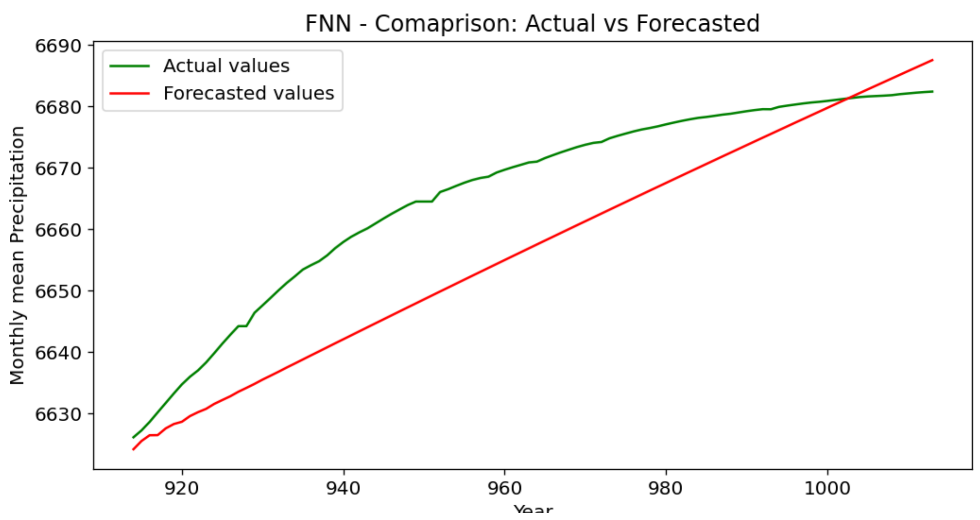


Figure 9: Forecast by ANN

4 Further Improvements

A further improvement that we suggest to our current model is to analyze the time series using a long short term memory RNN/ Since RNN's are able to remember past inputs and considering the improvements made by a feedforward neural network, we believe RNN could be used to further improve upon our results. Here the previous p time series data will be used to predict the current output

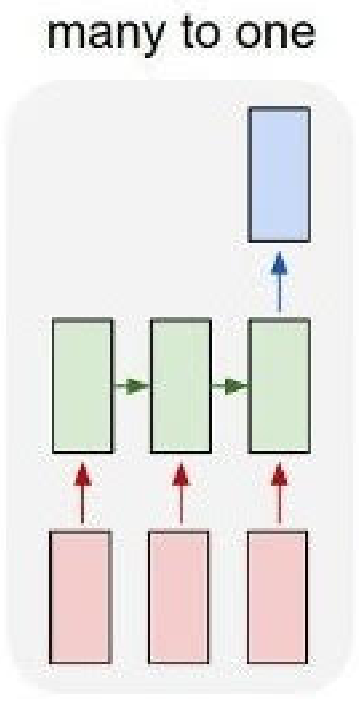


Figure 10: Recommended architecture for improvement

A Sleep Stage Scoring with Deep Learning

A.1 Introduction

This paper is primarily aimed at reducing the cost of monitoring sleep by using neural networks on data from a electroencephalography (EEG) instead of an overnight polysomnography (PSG). The latter requires highly trained physicians, and hence is expensive and time-consuming, compared to the former.
The task of this paper is to predict the sequence of sleep stages $S = \{s_1, s_2, \ldots, s_N\}$ based on the time-series data $X = \{x_1, x_2, \ldots, x_N\}$ as close as possible to $Y = \{y_1, y_2, \ldots, y_N\}$. Here x_i's are 30-second long epochs containing physiological signals and $y_i \in \{Wake, REM, N_1, N_2, N_3\}$ and each of these entries is a sleep stage with certain characteristics, and have been scored by well-trained technicians.
The physiological data is taken from the 153 recordings in the Sleep Cassette data, and 44 recordings from the Sleep Telemetry data. The data contains the features:

- Electroencephalography (EEG)
- Electrooculograapy(EOG)
- Chin Electromyography (EMG)

Each of which were sampled at 100 Hz for a period of 30 seconds. The data is first pre-processed by using the following pipeline:

1. Extract Fpz-Cz, Pz-Oz channels
2. Oversampling
3. Transform five consecutive 30-s input to 150-s input

For a comparative study, the author of the paper uses two models from literature:

1. CNN along with a Bi-LSTM
2. CNN

A.2 CNN + Bi-LSTM

This model contains two parts -

1. Representation Learning
2. Sequence Residual Learning

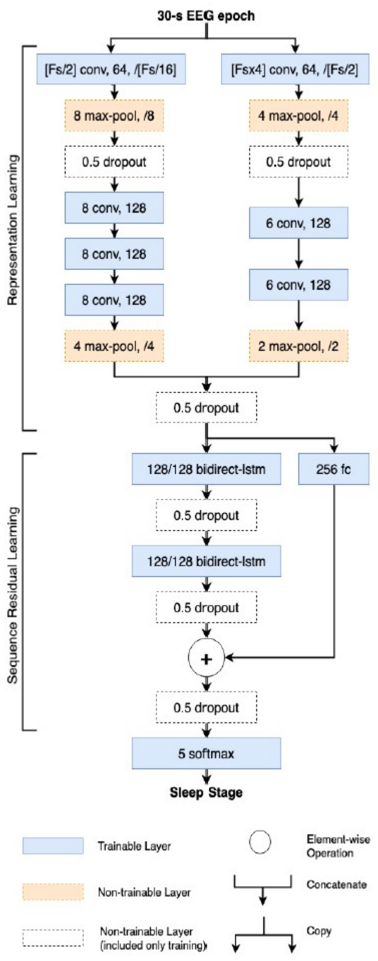


Figure 11: Summary of architecture used in Model 1

The representation learning is done by two convolutional networks in parallel. One with a large filter size, and one with a small one, to extract two different features - temporal and frequency precision. The small filter size captures temporal information, such as EEG patterns, while the large filter size efficiently captures the frequency information.
Sequence residual learning, on the other hand, is done by implementing a bi-directional Long short-term memory neural network (Bi-LSTM). this part of the network has been included to learn better, the patterns in temporal information, such as stage transition rules. Each Bi-LSTM has two LSTM processes to transmit data in forward and backward directions independently.
The final output is given by then concatenating the outputs of the two components and then using a fully connected layer with a softmax activation function to calculate

the probability of each sample belonging to a certain sleep class.

A.3 CNN

The CNN architecture comprises of two convolution and pooling layers each and three fully connected layers.

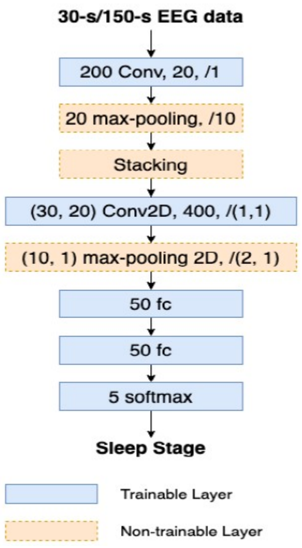


Figure 12: Summary of architecture used in Model 2

In contrast to the previous model, this architecture combines different frequency-specific trends over time to predict the class for every sample. The author uses the sparse categorical cross-entropy with L^2 regularization as the loss function for the model. ReLU is the activation function used to apply non-linearity to the model for the first two convolutional and dense layers; however, softmax is used to activate the last layer as this is a classification problem.

A.4 Results and Conclusion

The accuracy and the generalizability of the two models, results were used using 20-fold cross-validation. The following statistical tests across all folds as a metric for comparing performance:

- Mean of macro F1-score (MF1)
- Mean of overall accuracy (ACC)

- Cohen's Kappa Coefficient (κ)

The author reported that the first model took > 10 hours to finish running all 20 folds for 25-second epochs, and performed at an average accuracy of 75.4%, average MF1 of 67.8% and $\kappa = 0.66$ ($0.6 < \kappa < 0.8$) suggests that there is substantial agreement between the model and the data).
The second model, on the other hand, showed an average accuracy of 84%, after fold 5. MF1 score of the second model was 75%, and it obtained a κ score of 0.7. We can see that the second model performs better on all metrics; however, it takes substantially longer time than the first.
In conclusion, the author reasons that the reason for this significant difference in the accuracy of the two models was because of the limitations of their hardware (the author used 128 units for each Bi-LSTM as opposed to the 512 recommended by literature). The fact that the first model takes both frequency and temporal patterns into consideration, unlike the second model, which only looks for patterns in frequency, does not mean much, as the first model could not be implemented as literature recommended.

B Learning to Diagnose with LSTM RNNs

This paper used recurrent neural networks using Long-short term memory hidden units to recognize patterns in multivariate time series of clinical measurements. The data consisted of 10,401 pediatric intensive care units episodes, each a multivariate time series of 13 series diastolic and systolic blood pressure, peripheral capillary refill rate, end-tidal CO2, a fraction of inspired O2, Glasgow coma scale, blood glucose, heart rate, pH, respiratory rate, blood oxygen saturation, body temperature, and urine output. Potential uses of such data include classifying diagnoses accurately, predicting length of stay, predicting future illness and predicting mortality.
The proposed architecture consisted of LSTM RNN that uses memory cells with forget gates but without peephole connections. As output, they used a fully connected layer atop the highest LSTM layer followed by an element-wise sigmoid activation function, as the problem involved multi-label classification. Finally, a log loss function was used at each of the outputs. This architecture encountered the problem of having to pass the information across many sequence steps in order to affect the output. This problem was addressed by using sequential target replication, which replicated their sequence targets at every sequenced step providing a local error signal at each step. The loss is modified to be a convex combination of the final loss and the average loss overall steps. This method also helped the model to predict accurately even if the sequence was truncated by a small amount. Due to the large size of the data,

overfitting was a problem. This was dealt with by using L2 regularization and implementing judicious dropout. They reported that their best-performing LSTM, which used two layers of 128 memory cells, dropout of probability 0.5 between layers and target replication, outperformed the multi-layered perceptron with hand-engineered features. This was the first paper that used LSTMs to classify multivariate clinical data, and it produced promising results. This showed that the LSTMs could capture meaningful data in clinical data.

References

[1] https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9225319&tag=1

[2] https://github.com/thucdx/TSleep/blob/master/report/thucd2-sleep-project-final.pdf.

[3] https://arxiv.org/pdf/1511.03677.pdf

[4] Github Link: https://github.com/bharathvariar/DLProject