Prediction of Singapore’s Monthly Electricity Generation Capacity Using Time Series Deep Learning Techniques

Ryan Ueda Teo Shao Ming   
*Singapore Polytechnic*  
*School of Computing*Singapore  
ryanueda.21@ichat.sp.edu.sg

*Abstract*— Singapore's electricity generation has been steadily increasing over the years. This is due to rapid advancements in technology, allowing us to generate electricity more efficiently from various sources. Singapore's electricity generation data is also easily accessible and available via the singstat website, allowing us to analyze the data. In this paper we will explore and preprocess the data, plot decomposition plots, explore various LSTM network architectures and use time series deep learning techniques to attempt to predict the future of Singapore’s monthly total rainfall.

Keywords—explore; preprocess; deep learning; LSTM; time series; network architectures; decomposition plots; electricity

# **Introduction**

Recently, Singapore’s electricity generation capacity has been shown to be steadily increasing, with it being 289.6 GwH in February of 1975, and 4968.7 GwH in May of 2022. This is about a 1615% increase over the course of under 50 years, which is an exponential increase. EMA (Energy Market Authority) has taken steps in recent years to deploy equipment or technologies that are more energy efficient. As of today, 95% of Singapore’s electricity is generated using natural gas. Singapore is exploring ways to access regional power grids that are cost competitive. In addition, Singapore been hitting their targets for solar electricity production, and capitalising on emerging low-carbon alternatives. Although the general trend over the years seems to be increasing, there are dips in values quite often in the data. In this paper we aim to predict accurately the future of Singapore’s electricity generating capacity.

# **LSTMs, RNNs & Their Implementations**

**Fig. 1. LSTM Diagram**

Diagram, schematic

Description automatically generated

Similar to Recurrent Neural Networks (RNN), Long Short Term Memory (LSTM) models specialize in handling sequential data. Although RNNs are good at handling sequential data, they suffer from short-term memory. This means that if sequential data is long enough, the model has difficulty processing it. The cause is attributed to its vanishing gradient problem during back propagation, where the gradient shrinks to a possibly miniscule value as time passes.

However, LSTMs manage to solve this issue of short term memory. LSTMs are able to regulate the data flowing through the network using internal mechanisms known as ‘gates’. These gates are capable of learning what data in the sequence is of greater or lesser importance and then decide whether to keep the data or not. Variations of these gates include: Forget Gate, Input Gate and Output Gate.

**Fig. 2. LSTM Diagram**



Typically, in most LSTM networks, the sigmoid function is used as the gating function and tanh as the output activation function. Although tanh and sigmoid functions are relatively similar in terms of shape of the curve, the difference lies in their range of numbers. Sigmoid lies within 0 and 1, while tanh lies within -1 and 1

Combining these gate functionalities along with Sigmoid functions and cells and their state, LSTMs can be used in many real-world applications, such as state-of-the-art speech recognition, speech synthesis, language processing and text generation.

# **DATASET**

This dataset was obtained from the Singapore Government’s data statistics website – singstat.gov.sg. This dataset contains 569 rows and only two columns, Date and Electricity Generated. For every date, the electricity generated is presented in the adjacent column. The data is presented to us in a CSV (Comma Separated Values) file format. A sample of the dataset is shown below

**Fig 3 Attributes in singstat.gov.sg Dataset**

**Table

Description automatically generated**

Date: Presented in a Year-Month format for every month from 1975-01-01 to 2022-05-01

Electricity Generated: Amount of electricity generated per month, in gigawatts

Before proceeding, we performed some EDA (exploratory data analysis) on the dataset to identify any null, nonsensical or extreme (outliers) values that have to be taken care of to ensure that the performance of our model is not affected.

**Fig 4. Boxplot Of Electricity Generation**

Chart

Description automatically generated

From the boxplot, we can see that the highest value is about 5000, while the lowest is an extremely small number, less than 500. The mean electricity generated across the years is around 2500. There are no outliers in the data and the data range seems to be relatively reasonable.

**Fig 5. Number Of NULL In The Data**

**Graphical user interface, text, application, website

Description automatically generated**

A quick check shows us that there are no null values in the dataset that have to be imputed.

**Fig 6. Electricity Generation Capacity from 1975-2022**

Chart, line chart

Description automatically generated

From the line graph plotted above, we can see that Singapore’s electricity generation capacity increases. Around the third quartile of the year however, electricity generation seems to dip slightly, before increasing again.

# **METHODOLOGY**

After performing EDA (Exploratory Data Analysis), we would clean the dataset and impute the values. However, this dataset does not contain any null values or irregular data such as outliers. Hence, we do not have to perform any data pre-processing on this dataset other than changing the index of the dataframe to be Date. We will pre-process the data for it to be ready to be fitted into the models.

**Fig 7. Electricity Generation Over The Years**

Chart

Description automatically generated

From the run-sequence plot of the time series above, we can see that there is a very strong positive trend in the data and is very likely to be non-stationary. We can confirm this using a decomposition plot, as well as checking the means/variances of the data along with the ADF (Augmented Dickey-Fuller) test to check for stationarity

**Fig 8. Decomposition Plot**

A picture containing graphical user interface

Description automatically generated

Based on the decomposition plot above, we can visually conclude that there is a clear positive trend in the data, and some form of seasonality. This can be confirmed by a quick check.

**Fig 9. Mean and Variance Check**





A quick check on the means and variances of the data shows that they have both drastically different means as well as variances, implying that the data is not stationary.

**Fig 10. Mean and Variance Check**

Graphical user interface, text, application

Description automatically generated

To confirm this, we will perform the ADF (Augmented Dickey-Fuller) test.

Upon conducting this test, it returns a p-value of 0.990521, which is extremely high. This is sufficient evidence to show that we cannot reject the nulll hypothesis that the data is stationary. Hence, we can conclude that the time series is not stationary. However, since LSTM networks are able to learn non-linear and non-stationary natures of a time series, we will not touch the data.

**Fig. 11 Train & Test Sets**

Graphical user interface, text

Description automatically generated

To form the train and test sets, we split the data for the train set to have the first 400 samples of data, and the test set to have 169 samples. We then normalize the data using MinMaxScaler.

**Fig. 12 Time Series To Input & Labels**

Text

Description automatically generated

We then convert our time series data into inputs and labels

**Fig. 13 Reshape Data**

Text

Description automatically generated

Next, we reshape the data in the above format in order to allow for the data to fit into our model.

**Fig. 14 Base Model Architecture**

Table

Description automatically generated

Now that the time series data is ready to be fitted into our mdoel, we create a simple baseline LSTM model. It starts with an input layer of 32 neurons, followed by a Dense layer with 32 neurons and a relu activation. Finally, its passed into the final Dense layer with a single neuron, representing our output. We then compile the model with the adam optimizer, with a learning rate of 0.01 and the mean squared error loss metric. This returns us a model with a total of 5,697 parameters

**Fig. 15 Training The Model**

Table

Description automatically generated

Upon training the model for 300 epochs and a batch size of 64, we can see towards the last few epochs that the loss is around 0.44, which is not exactly ideal. Nonetheless, lets visualise the results to better understand the performance of our model.

**Fig 16. Train & Test RMSE**

Text

Description automatically generated with medium confidence

After training the model, we then predict and inverse transform our data to its original form, before calculating the root mean squared error of our train and test sets. We can see that the model performed very poorly, obtaining extremely high RMSE scores for both train and test, at 1291 and 913 respectively.

**Fig 17. Train: Actual vs Pred Plot**

Chart

Description automatically generated

By plotting the predicted train and test values, we can gain a better visualisation and understanding of the performance of our model. We can see that the predicted values in red is a flat straight line, which is not unexpected after looking at the RMSE of our model.

**Fig 18. LSTM Model 2**

Table

Description automatically generated

The poor performance of the baseline model is likely to be attributed to the the small and shallow nature of the network. Hence, we will increase the size of this network to see if our model will improve in any way.

This model follows the same basis as the previous model, but now includes an additional LSTM layer with 64 neurons. We then add a Flatten layer before passing it to a new Dense layer with 64 neurons. In addition, a relu activation is added to each of the Dense layers. To prevent overfitting, a L2 kernel regularizer with a penalty of 0.05 is added to the Dense layers. In addition, a dropout layer with a rate of 0.3 is passed between the Dense layers. This returns us a model with a total of 35,713 parameters.

**Fig. 19 Fitting Model 2**

**Table

Description automatically generated with medium confidence**

In the new model, we can see that towards the last few epochs, the loss

is much better as compared to before at around 0.095.

## **Fig. 20 Model 2 Train & Test RMSE**



This time, the model returns a significantly lower RMSE score for both train and test sets at 90 and 178 respectively. Let’s visualise these results to see how much our model has improved.

## **Fig. 21 Train: Actual vs Pred Plot**

Chart

Description automatically generated

From the plot above, we can see that the model has improved quite significantly. The red predicted lie seems the follow the rough trend of the blue line indicating the actual values of the time series. Although it follows the trend quite well, it does not converge with the actual values as well, and has room for improvement.

# **Model Improvement**

Now that we have obtained a decently good model for our time series forecasting task, we can proceed to improve the model further. This can be done by hyperparameter tuning.

**Fig. 22. Hyper Parameter Tuning**

**Text

Description automatically generated**

To perform hyperparameter tuning, we will be using keras tuner to tune the following parameters: kernel\_regularizer, dropout rate, optimizer, and activation.

**Fig. 23. Hyper Parameter Tuning**

Text

Description automatically generated

After running our hyperparameter tuning, keras tuner returns us the best parameters for our model.

The best parameters for our model are: relu activation, l2 kernel regularizer, a dropout rate of 0.1 and the adam optimizer.

**Fig. 24. Hyperparameter Model**

Table

Description automatically generated

The model architecture of the hypertuned model is relatively the same as the previous model, but now with a different dropout value.

In theory, this model will perform better than the previous model. To check its reliability, lets test the model.

**Fig. 25 Fitting The Hypertuned model**

A picture containing table

Description automatically generated

Upon fitting the model, we can see that by the end of the 400 epochs, we are at a mean squared error of 0.0018, which far better than before. To understand the differences, lets visualise the results.

**Fig. 26 Hypertuned Model Train & Test RMSE**



The train and test RMSE has once again improved, with scores of 57 and 177 respectively. This shows that the hypertuning managed to yield satisfactory results.

**Fig. 27 Train: Actual vs Pred Plot**

**Chart, line chart

Description automatically generated**

From the line plot above, we can see that the red predicted values match the blue actual values very well and manage to capture the trend very impressively as well. This is an improvement from the previous models, and a huge difference from our baseline model.

**Fig. 28 Train: Actual vs Pred Plot**

**Chart, line chart

Description automatically generated**

Now that we have achieved a model up to our standards, we can predict on the test data. From the line graph above, we can see that although the red predicted values do not match up exactly to the yellow actual values of the test set, it follows the trend of the time series very well and manages to capture a large majority of variance in the series.

# **CONCLUSION**

In this paper, we have discussed how to analyze and forecast the electricity generation capacity of Singapore along with various deep learning techniques. These include how to perform exploratory data analysis on time series data, preprocess and prepare data for time series analysis using LSTMs, checking for stationarity, how to create LSTM model architectures, as well as how to perform hyperparameter tuning on our model.

# **DATASET LINK**

<https://tablebuilder.singstat.gov.sg/table/TS/M890831>