PROJECT

REPORT

TIME SERIES FORECASTING (SUNSPOTS)

SUBMITTED BY: RATNESH DUBEY (UE178074)

PRAYANSH PANDEY (UE178068)

MD SHAHNAWAZ ALAM (UE178058)

BRANCH - IT - SEM 8TH

SUBMITTED TO: Dr. VEENU MANGAT

TABLE OF CONTENTS

INTRODUCTION

The sunspot numbers are the major targets which describes the solar activity level. The solar activity influences the human health and living environment on earth. Long-term prediction of sunspot activity can provide important reference information for aerospace, communication, disaster prevention, and so on.

Therefore, the research of sunspot number time-series prediction method has been an important and hot topic in this field. The data of annual and monthly sunspot activity is considered as typical nonlinear, non-Gaussian, and nonstationary time series, which has obvious characteristics of chaotic sequences. The accurate prediction of sunspot numbers has great research significance. It is difficult to predict using the traditional method and the accuracy is not high. [4] [5]

Time series prediction (forecasting) has experienced dramatic improvements in predictive accuracy as a result of the data science machine learning and deep learning evolution. As these ML/DL tools have evolved, businesses and financial institutions are now able to forecast better by applying these new technologies to solve old problems. In this article, we showcase the use of a special type of **Deep Learning model called an LSTM (Long Short-Term Memory)**, which is useful for problems involving sequences with autocorrelation. We analyze a famous historical data set called "sunspots" (a sunspot is a solar phenomenon wherein a dark spot forms on the surface of the sun). We'll show you how you can use an LSTM model to predict sunspots 11 years into the future with an LSTM model.

We have also implemented some other models to compare their performance with LSTM model.

MOTIVATION

The variability of solar activity over time demonstrates highly complex behavior. Neural networks provide useful models for predicting that behavior. Accurate predictions of solar activity are essential for satellite tracking and constellation management, orbit lifetime studies, and mission

planning for long-term orbiting platforms, such as the proposed space station Freedom.

OBJECTIVE

The aim of this project was to develop a Neural Network model for forecasting sunspot activity accurately, we also want to predict the behavior of SC 25. The simplicity and recent success of Deep Neural Networks in forecasting solar-terrestrial phenomena explain why this method has been chosen for the SC 25 prediction.

MODEL COMPONETS

1. Neural Networks
2. Convolution Layers
3. RNN Layers
4. LSTM Layers
5. Lambda Layers
6. Callbacks
7. Dense Layers.

TECHNOLOGY USED

TensorFlow

DEPENDENCIES

- Internet Connection
- GPU(Graphical Processing Unit)
- Labelled Data.
- External Libraries

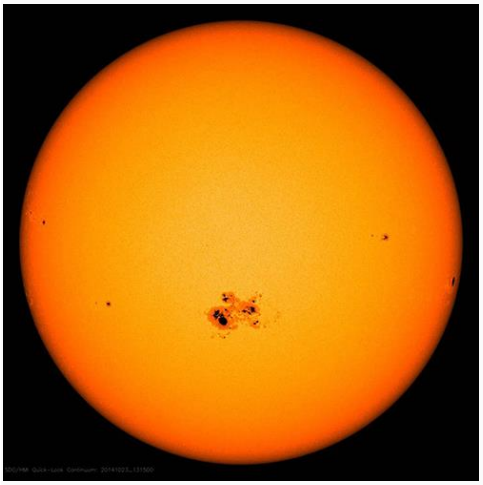
DATA SOURCES

- Kaggle Dataset (Link: https://www.kaggle.com/robervalt/sunspots)

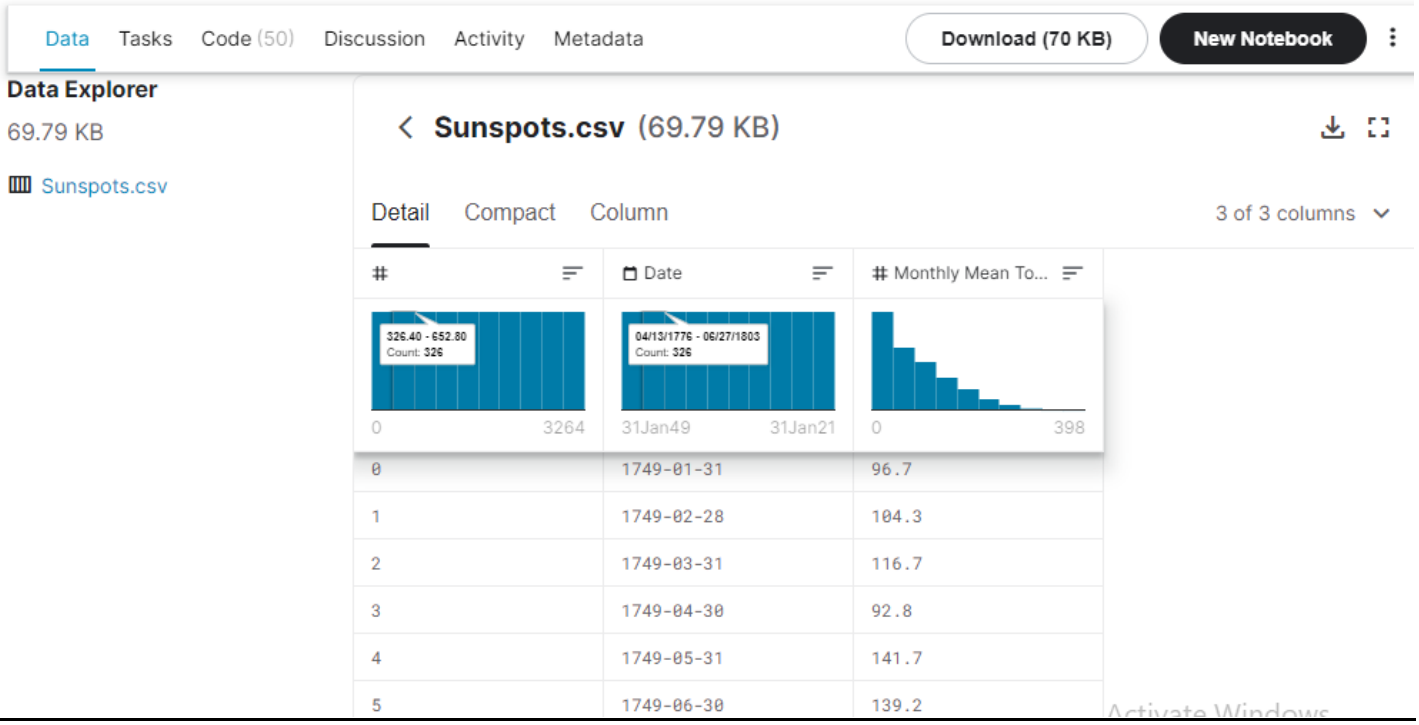
DATA OVERVIEW

Sunspots is a famous data set that ships with R (refer to the

Datasets library). The dataset tracks sunspots, which are the occurrence of a dark spot on the sun. Here's an image from NASA showing the solar phenomenon. Pretty cool!

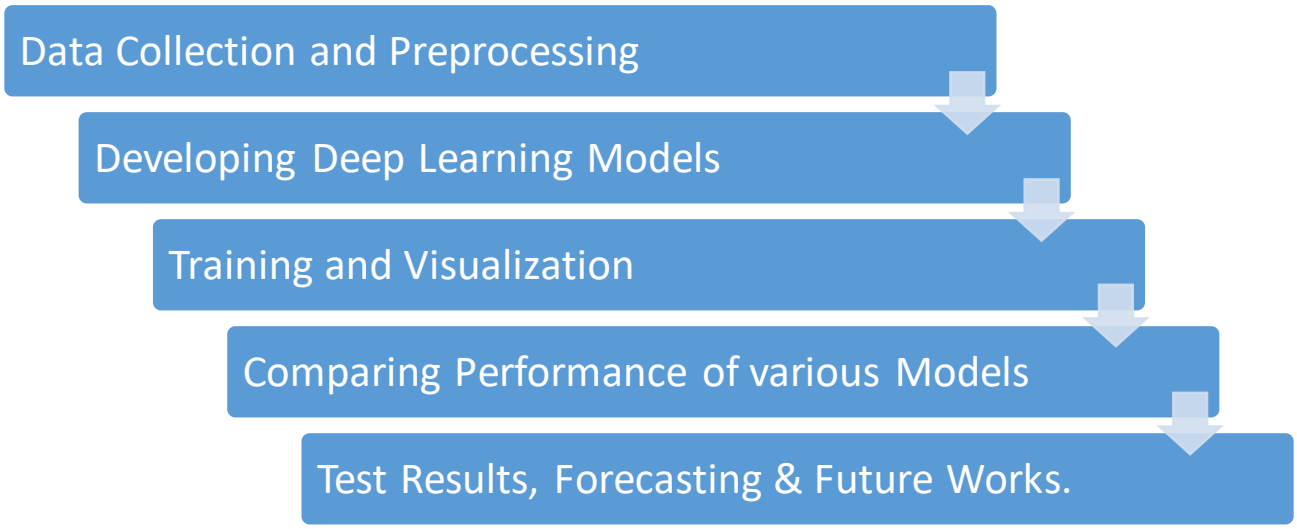


The dataset is a CSV file which contains 3 columns and 3236 rows , 1st column is the serial number, 2nd is date and the third is Monthly Mean Total Sunspot Number.



PROPOSED METHODOLOGY

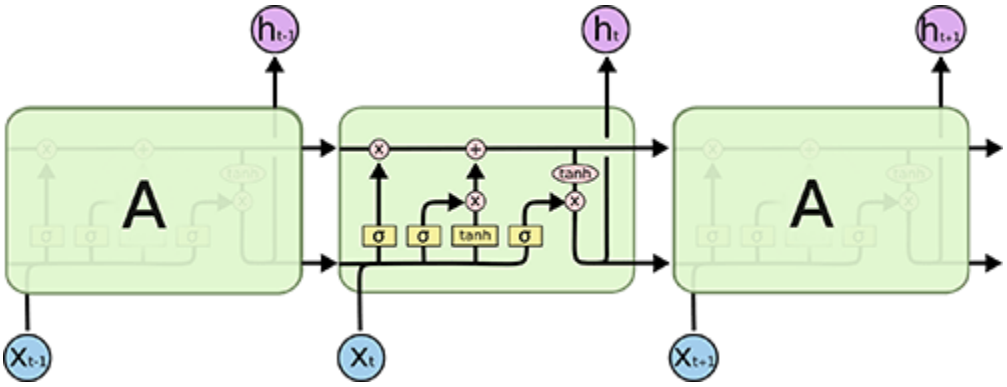
The methodology will contain the following steps:



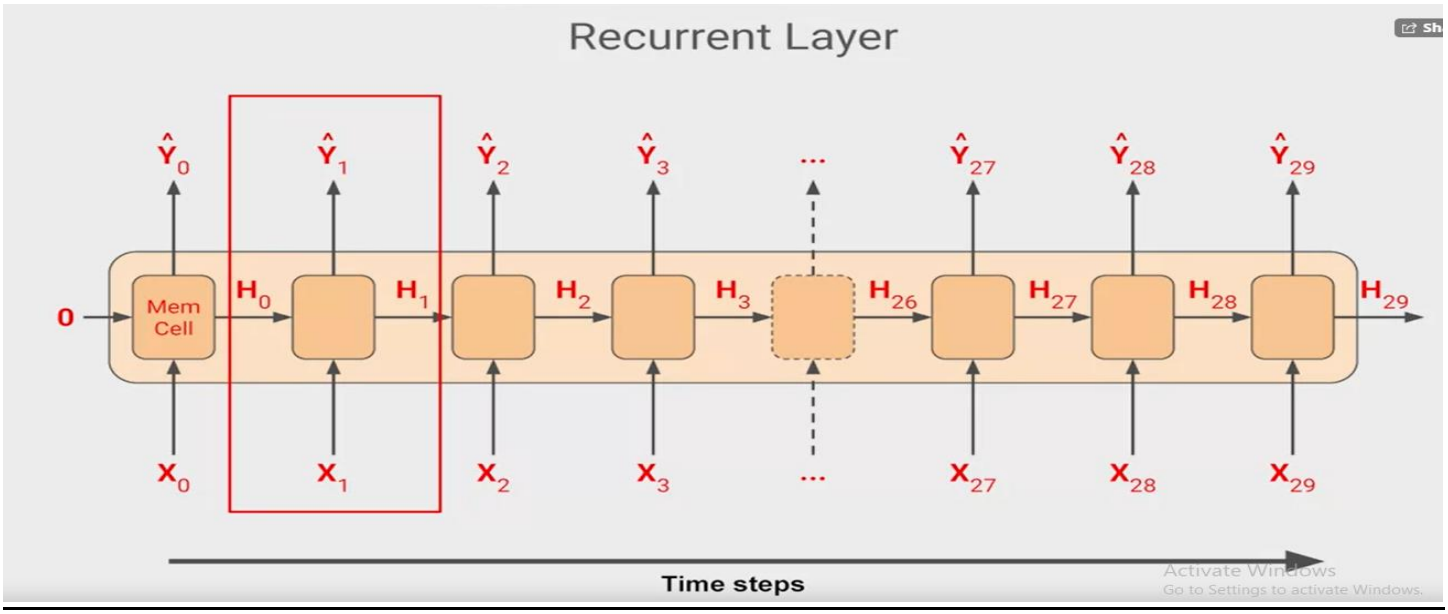
WORK PLAN

1. First we will import the Sunspots data set and perform some preprocessing steps on it.
2. After importing and preprocessing of data, we will do some exploratory data analysis on the data, we try to plot some graphs to see how data trends over time.
3. Create train/test/validation dataset from original data.
4. After that we will apply different models on the data, these contain naïve approach, Moving Averages, Linear Regression, RNNs, and LSTM & Convolutions.
5. We can also add some dense and lambda layers to the model for multilayer Neural Network training, normalization and scaling purpose.
6. Plotting and predicting the results and calculating the mean absolute error.
7. We will then compare the performances of the various models.
8. Finally whole project will be implemented in the form of interface system for users.

A simple LSTM network.



LSTM are just like RNNS [8]



OBJECTIVES ACHIEVED

1. Six Models are created.
2. Performance and loss is compared
3. Graphs are plotted.
4. Prediction is made.
5. LSTM combined with Convolutions seems to perform best with this data.

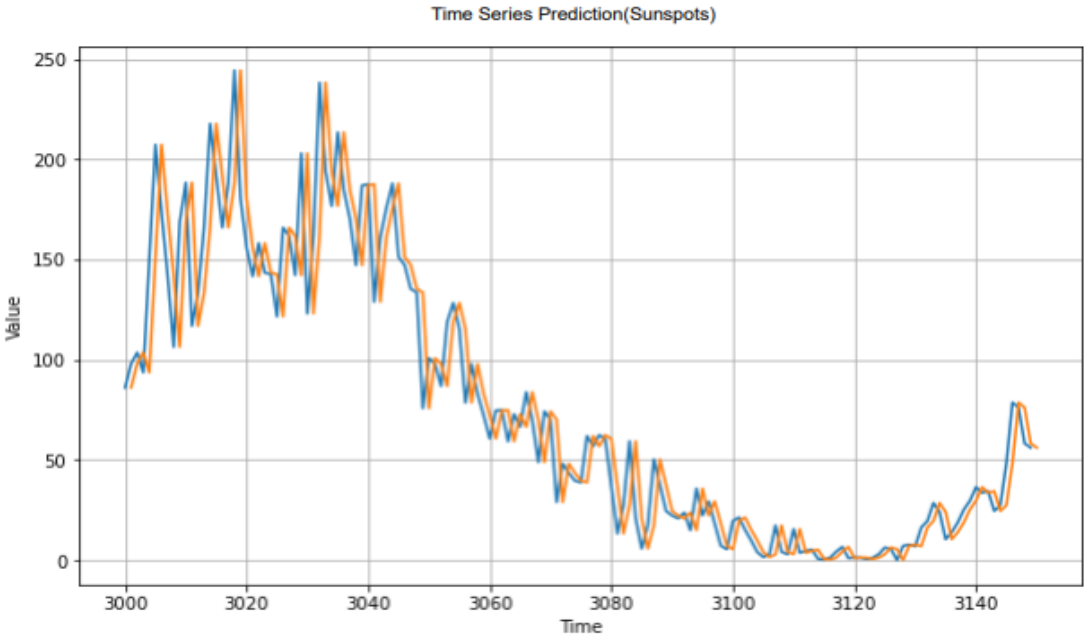
RESULTS

We have used 6 different models to forecast our Time Series, and in each model we compute the mean absolute error, as well as plotted the graph to better understand how well the model is performing.

Naïve Approach [2] [8]

In Naïve Approach we used the most basic approach of using the previous outputs as the future outputs.

V (t) = V (t-1) //here t is current time, and t-1 is one time step before current time.



Here we got the Mean Absolute Error of around 16.14, we will take this a baseline.

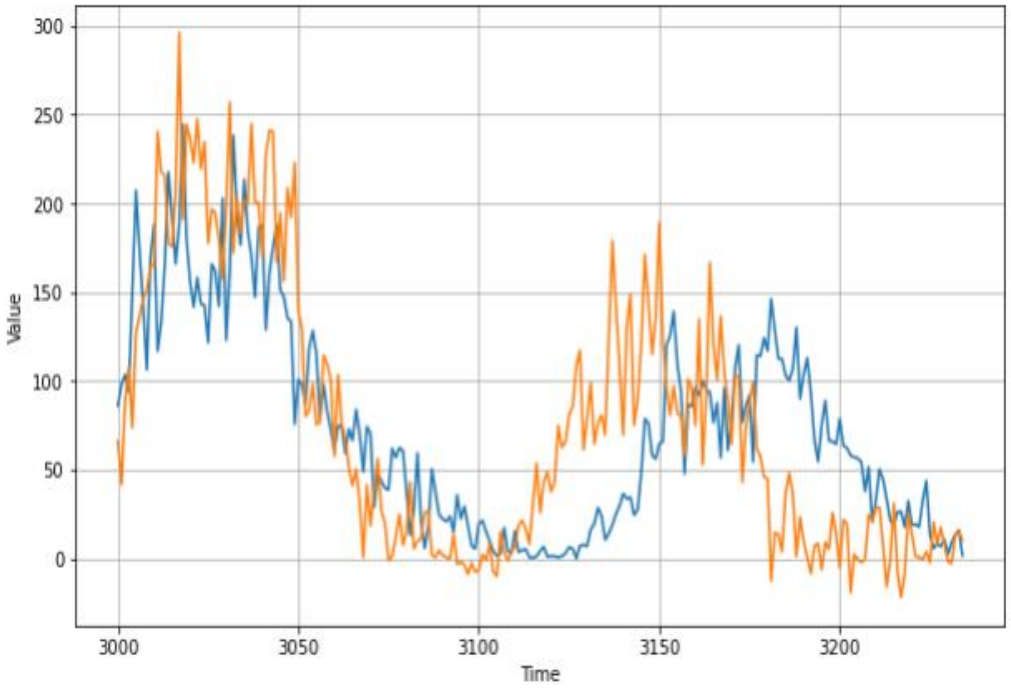
Moving Averages & Differencing [2] [8]

In statistics, a moving average is a calculation used to analyze data points by creating a series of averages of different subsets of the full data set.

In layman terms Moving Averages is a statistical technique in which values are averaged over a period of time to predict future values at particular time.

Moving averages can be used to reduce the noise in the data but it does not consider seasonality & trend, so as to prevent this, we first use differencing on the series to remove seasonality and trend, this can be done by subtracting the value at some previous time from current time.

After performing this step, we apply moving averages on the series and then again add the subtracted values in the series to get our forecasted series.



Mean Absolute and Mean Sqaure Error for Moving Averages& Differencing

```
print(tf.keras.metrics.mean_squared_error(x_valid, diff_moving_avg_plus_past).numpy())
print(tf.keras.metrics.mean_absolute_error(x_valid, diff_moving_avg_plus_past).numpy())
```

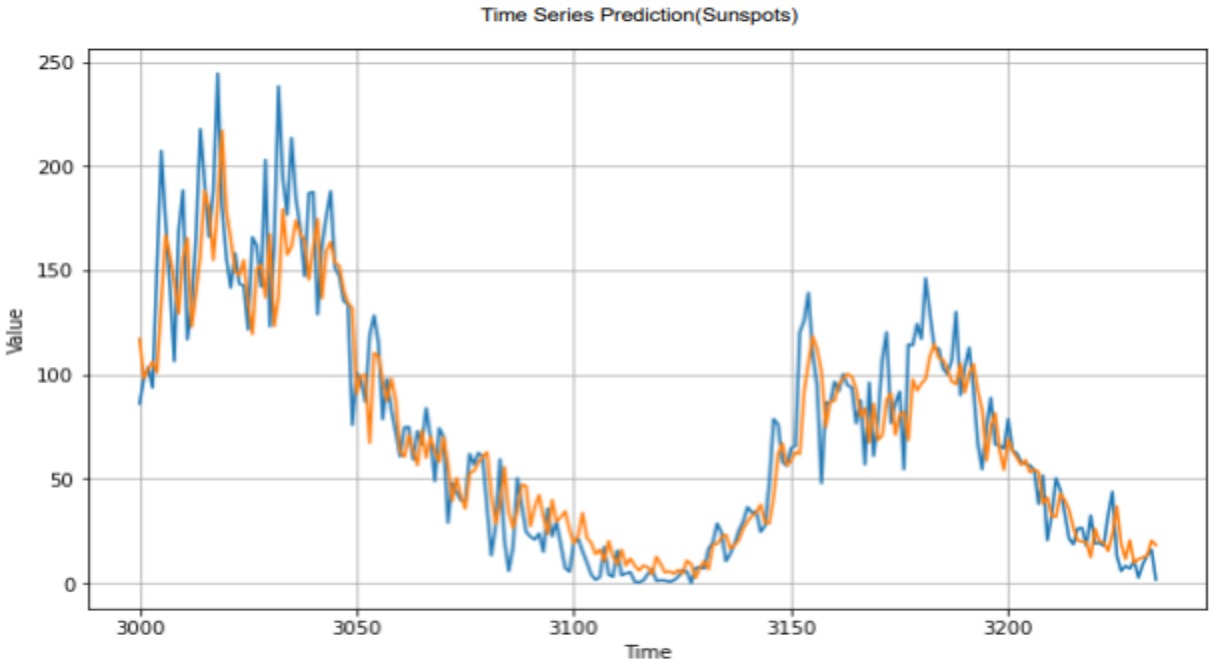
```
2976.2167495035455
43.33463829787234
```

Forecasting Using Linear Regression [10]

We can create a Linear Regression model by using a single neuron Deep Neural Network.

The input shape we fed into the neural network will be equal to window size, the output will be a single value.

We can also see the resultant weights and the bias value by using get_weights () method.



Mean Absoule Errror for Linear Regression(single neuron)

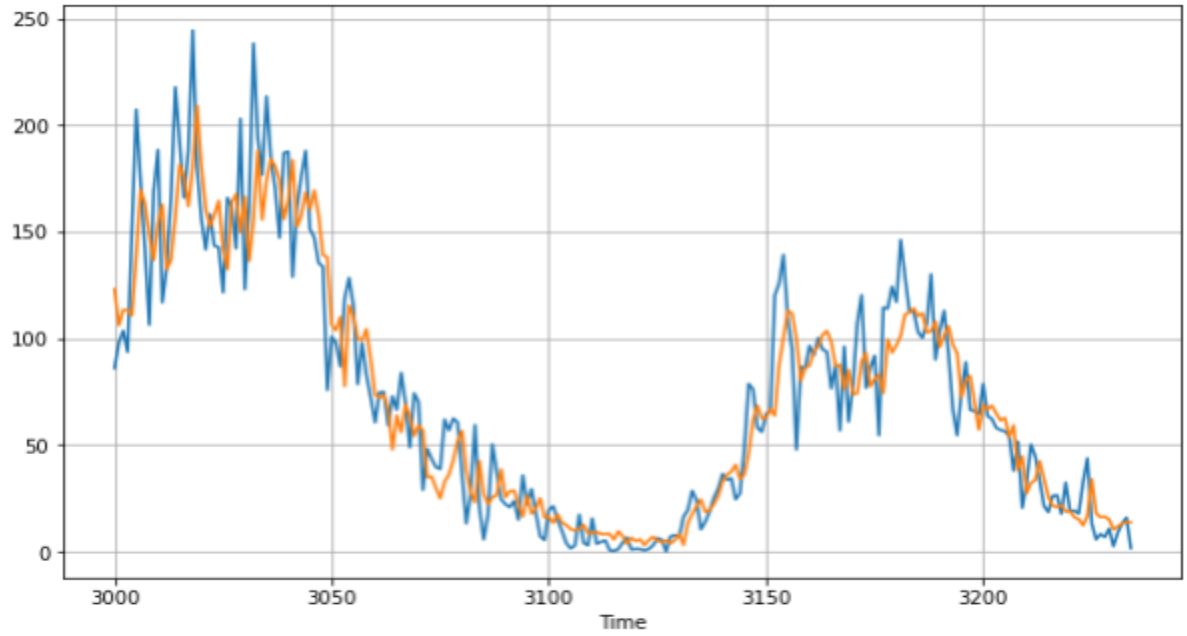
```
tf.keras.metrics.mean_absolute_error(x_valid, results).numpy()
```

```
15.3134165
```

This is the least mean absolute error until now, so from this we can conclude that Linear Regression seems to perform better the Naïve & Moving Averages approach.

Forecasting Using DNN (Deep Neural Networks) [6]

Creating a DNN model is similar to creating a Linear Regression model, the difference is that we will be using multiple layers also called DNN layers in our model, for this project we will be implementing 3 layer mode, 2 DNN layers and 1 output layer, We will be using **relu** activation function.



Mean Square Error.

```
tf.keras.metrics.mean_absolute_error(x_valid, results).numpy()
```

```
15.176499
```

Forecasting Using RNNs & Convolutions [6] [8]

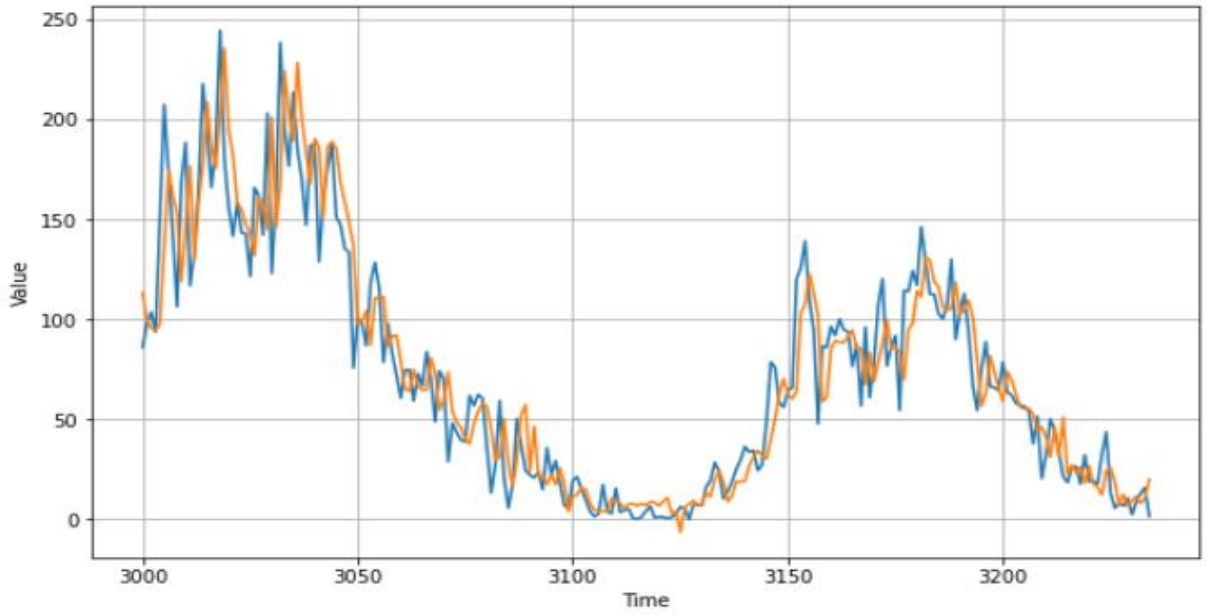
A recurrent neural network (RNN) is a type of artificial neural network which uses sequential data or time series data.

Like feedforward and convolutional neural networks (CNNs), recurrent neural networks utilize training data to learn. They are distinguished by their "memory" as they take information from prior inputs to influence the current input and output. [9]

In our model we have used 2 RNN layers along with convolution layer to forecast our time series.

We'll first run for 100 epochs and determine optimal learning rate for our stochastic gradient decent (SGD), and then we retrain our model for 500 epochs to get our **MAE** (**Mean Absolute Error**) and predictions.

RNNs have a memory, but the effect of earlier inputs diminish over time in RNNs unlike LSTMs.



Mean Square Error.

```
# MAE
tf.keras.metrics.mean_absolute_error(x_valid, rnn_forecast).numpy()
```

```
15.583032
```

Forecasting Using LSTM & Convolutions [6] [8] [11]

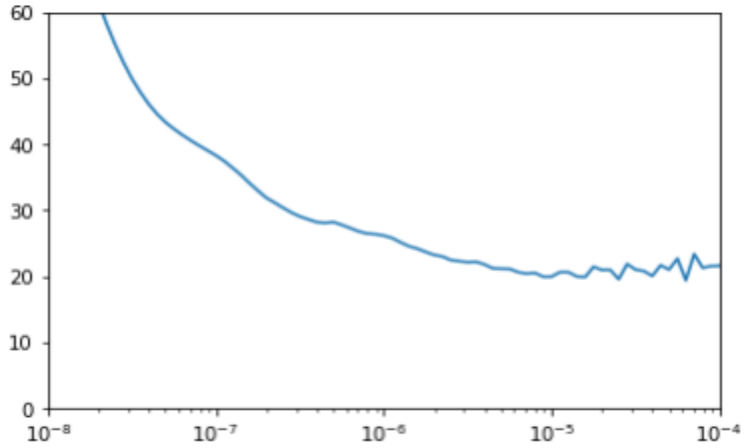
For our final model we will be using convolutions along with LSTMS, this helps us to create a model that fits really well to our data, we can also use various tuning methods to further improve our performance.

LSTMS also known as Long Short Term Memory, It is a type of RNN(Recurrent Neural Network) which helps us preserve the information about current state throughout the network, and thus this helps us make better predictions as recent past values have more impact on future values.

In this model, we used one convolution layer with 32 filters, after that we have added 2 LSTM layers, after that we have added 3 dense neuron layers, with 30, 10 and 1 units respectively.

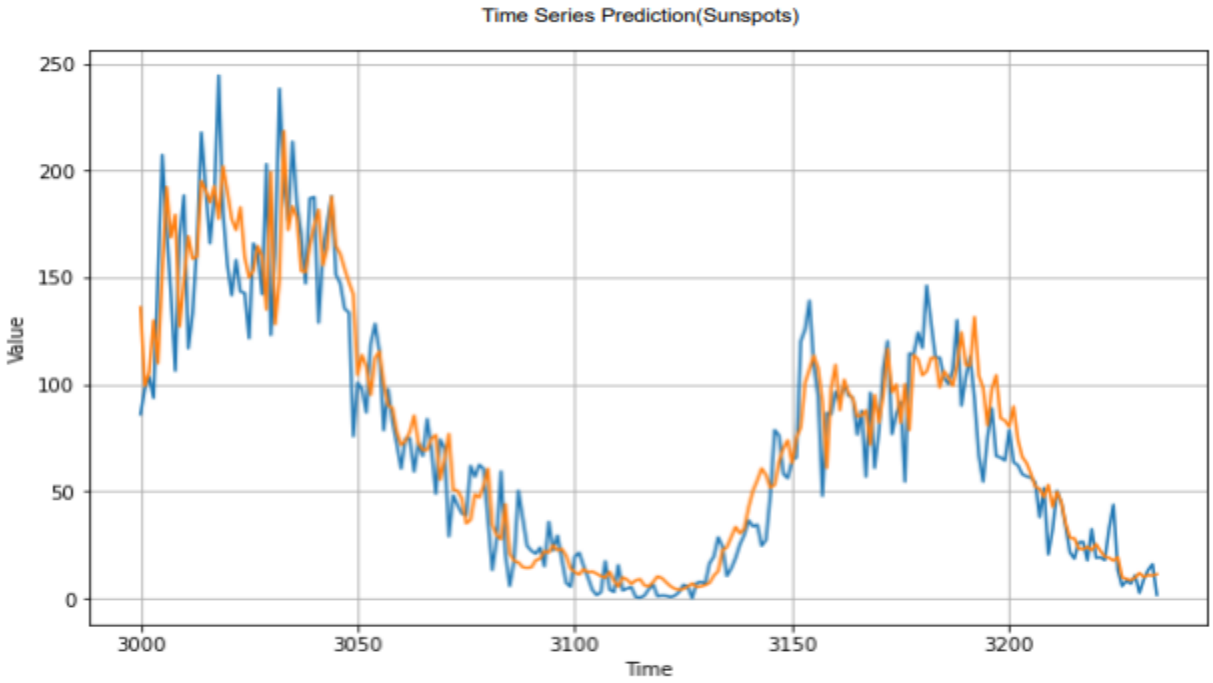
First we train the model for 100 epochs to get an optimal learning rate for our stochastic gradient decent (SGD) optimizer.

```
(1e-08, 0.0001, 0.0, 60.0)
```



After getting the optimal learning rate, we retrain the model for 500 epochs with our optimal learning rate.

After creating the model, we will create a forecast array of predicted values, and then we plot our results.

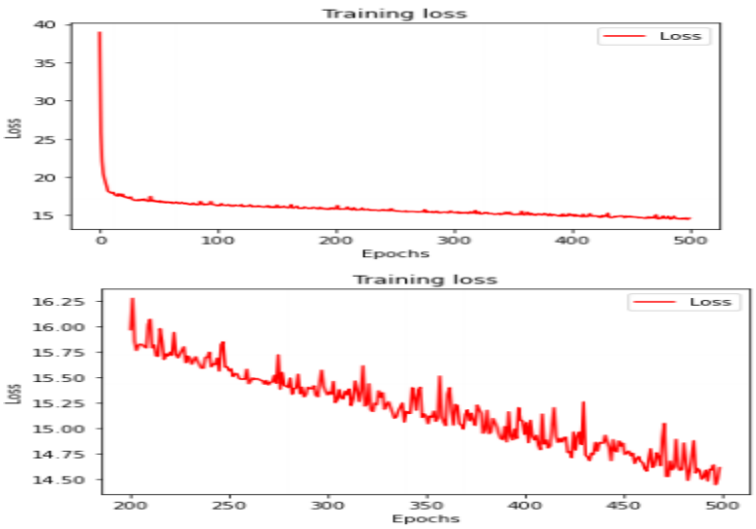


Mean Square Error.

```
tf.keras.metrics.mean_absolute_error(x_valid, rnn_forecast).numpy()
```

```
15.089574
```

Training Loss vs Epochs.



Forecasted Values.

```
print(rnn_forecast)
```

```
[135.92502    99.271904  105.50781   129.8834    109.807526  150.31667
 192.19649   168.62233   179.26598   126.94186   148.1779    169.16774
 158.67053   159.65434   195.03677   190.81548   185.02423   192.65051
 177.28357   201.87624   190.02647   177.29008   172.10686   182.73575
 159.40915   149.92252   153.19821   164.56311   159.92856   134.9303
 199.1364    128.0686    148.98816   218.37451   172.10358   183.2
 178.08499   152.958     152.39517   164.84302   174.30031   181.5857
 155.59616   163.76837   187.73816   164.62791   160.91759   153.94258
 147.61807   142.00415   104.43032   113.7124    108.309616   95.00554
 112.304596  115.40711   100.481155   90.23153    88.875725   77.811775
```

.....

As we see that **Mean absolute Error** is quite low, thus indicating that with little more tuning, we can predict sunspots activity quite accurately.

LSTMs performance turns out to be better the RNNs, contributing to the fact that LSTMs retains the effect of earlier inputs throughout the state while RNNs do not.

CODE (LSTM AND CONVOLUTIONS)

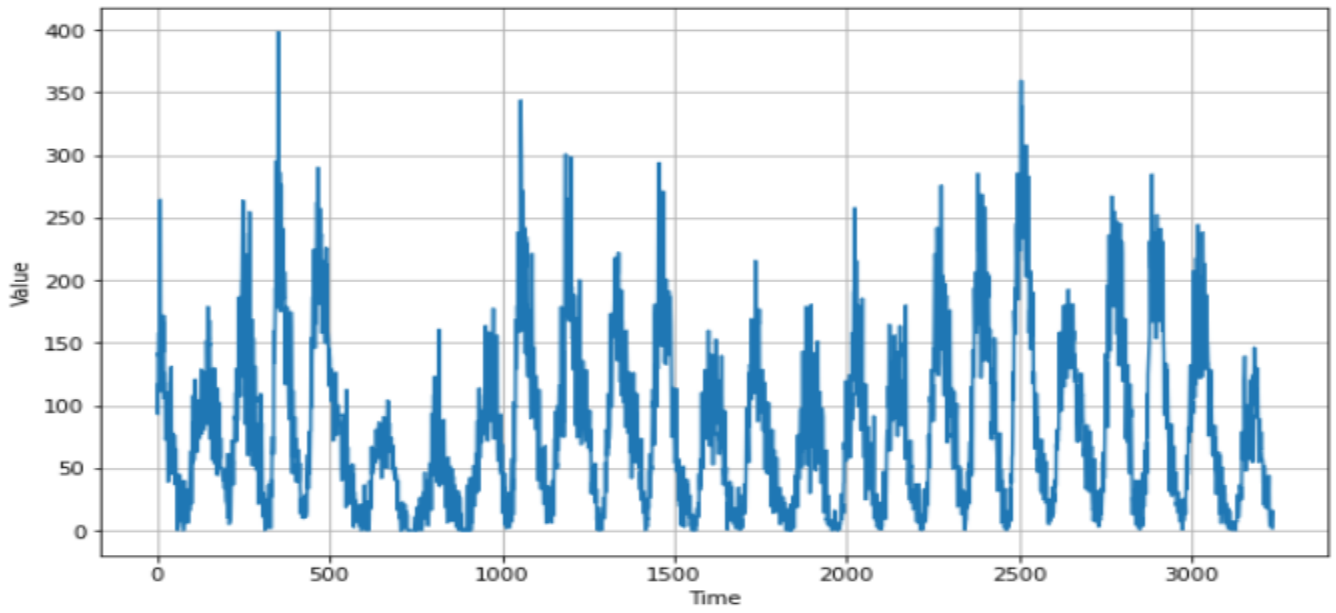
```
import tensorflow as tf
print(tf.__version__)


import numpy as np
import matplotlib.pyplot as plt
def plot_series(time, series, format="-", start=0, end=None):
    plt.plot(time[start:end], series[start:end], format)
    plt.xlabel("Time")
    plt.ylabel("Value")
    plt.grid(True)
```

```
2.3.0
```

Importing Data CSV file and visulaizing

```
import csv
time_step = []
sunspots = []

with open('Sunspots.csv') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    next(reader)
    for row in reader:
        sunspots.append(float(row[2]))
        time_step.append(int(row[0]))
```

```
series = np.array(sunspots)
time = np.array(time_step)
plt.figure(figsize=(10, 6))
plot_series(time, series)
```



```
#Creating new windowsize, batch size and shuffle buffer size
split_time = 3000
time_train = time[:split_time]
x_train = series[:split_time]
time_valid = time[split_time:]
x_valid = series[split_time:]

window_size = 30
batch_size = 32
shuffle_buffer_size = 1000
```

```
#Converting the dataset into windowed dataset
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
    series = tf.expand_dims(series, axis=-1)
    ds = tf.data.Dataset.from_tensor_slices(series)
    ds = ds.window(window_size + 1, shift=1, drop_remainder=True)
    ds = ds.flat_map(lambda w: w.batch(window_size + 1))
    ds = ds.shuffle(shuffle_buffer)
    ds = ds.map(lambda w: (w[:-1], w[1:]))
    return ds.batch(batch_size).prefetch(1)
```

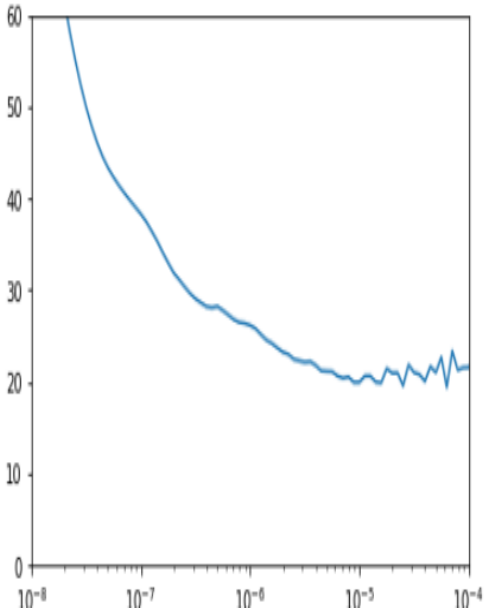
```
#Helper Function for forecasting the data.
def model_forecast(model, series, window_size):
    ds = tf.data.Dataset.from_tensor_slices(series)
    ds = ds.window(window_size, shift=1, drop_remainder=True)
    ds = ds.flat_map(lambda w: w.batch(window_size))
    ds = ds.batch(32).prefetch(1)
    forecast = model.predict(ds)
    return forecast
```

```
tf.keras.backend.clear_session()
window_size = 64
batch_size = 256
train_set = windowed_dataset(x_train, window_size, batch_size, shuffle_buffer_size)
print(train_set)
print(x_train.shape)

model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=32, kernel_size=5,
                      strides=1, padding="causal",
                      activation="relu",
                      input_shape=[None, 1]),
  tf.keras.layers.LSTM(64, return_sequences=True),
  tf.keras.layers.LSTM(64, return_sequences=True),
  tf.keras.layers.Dense(30, activation="relu"),
  tf.keras.layers.Dense(10, activation="relu"),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 400)
])

lr_schedule = tf.keras.callbacks.LearningRateScheduler(
    lambda epoch: 1e-8 * 10**(epoch / 20))
optimizer = tf.keras.optimizers.SGD(lr=1e-8, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])
history = model.fit(train_set, epochs=100, callbacks=[lr_schedule])
```

```
WARNING:tensorflow:AutoGraph could not transform <function windowed_dataset.<locals>.<lambda> at 0x0000022F7C483040> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full o
utput.
Cause: module 'gast' has no attribute 'Index'
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function windowed_dataset.<locals>.<lambda> at 0x0000022F7C483040> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full o
utput.
Cause: module 'gast' has no attribute 'Index'
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
<PrefetchDataset shapes: ((None, None, 1), (None, None, 1)), types: (tf.float64, tf.float64)>
(3000,)
Epoch 1/100
12/12 [==============================] - 2s 199ms/step - loss: 79.8340 - mae: 80.3314
Epoch 2/100
12/12 [==============================] - 2s 181ms/step - loss: 78.0944 - mae: 78.5918
Epoch 3/100
12/12 [==============================] - 2s 185ms/step - loss: 75.4519 - mae: 75.9497
Epoch 4/100
12/12 [==============================] - 2s 188ms/step - loss: 72.2678 - mae: 72.7658
Epoch 5/100
12/12 [==============================] - 2s 183ms/step - loss: 68.7693 - mae: 69.2672
Epoch 6/100
```

```
plt.semilogx(history.history["lr"], history.history["loss"])
plt.axis([1e-8, 1e-4, 0, 60])
```

```
(1e-08, 0.0001, 0.0, 60.0)
```

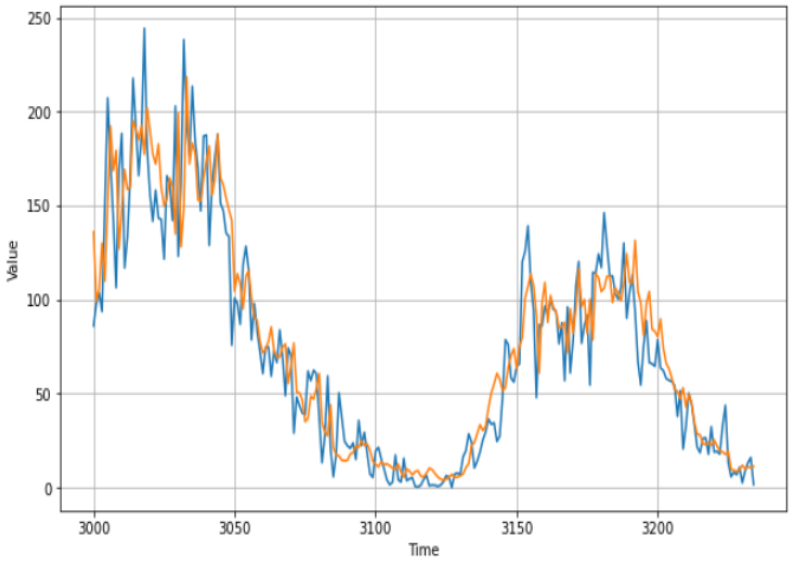


```
tf.keras.backend.clear_session()
train_set = windowed_dataset(x_train, window_size=60, batch_size=100, shuffle_buffer=shuffle_buffer_size)
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=60, kernel_size=5,
                      strides=1, padding="causal",
                      activation="relu",
                      input_shape=[None, 1]),
  tf.keras.layers.LSTM(60, return_sequences=True),
  tf.keras.layers.LSTM(60, return_sequences=True),
  tf.keras.layers.Dense(30, activation="relu"),
  tf.keras.layers.Dense(10, activation="relu"),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 400)
])


optimizer = tf.keras.optimizers.SGD(lr=1e-5, momentum=0.9)
model.compile(loss=tf.keras.losses.Huber(),
              optimizer=optimizer,
              metrics=["mae"])
history = model.fit(train_set,epochs=500)
```

```
WARNING:tensorflow:AutoGraph could not transform <function windowed_dataset.<locals>.<lambda> at 0x0000022F0E910280> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full o
utput.
Cause: module 'gast' has no attribute 'Index'
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function windowed_dataset.<locals>.<lambda> at 0x0000022F0E910280> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full o
utput.
Cause: module 'gast' has no attribute 'Index'
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
Epoch 1/500
30/30 [==============================] - 4s 137ms/step - loss: 38.9182 - mae: 39.4146
Epoch 2/500
30/30 [==============================] - 4s 119ms/step - loss: 25.7634 - mae: 26.2578
Epoch 3/500
30/30 [==============================] - 4s 123ms/step - loss: 22.0849 - mae: 22.5786
Epoch 4/500
30/30 [==============================] - 3s 112ms/step - loss: 20.4691 - mae: 20.9621
Epoch 5/500
30/30 [==============================] - 4s 123ms/step - loss: 19.7838 - mae: 20.2763
Epoch 6/500
30/30 [==============================] - 4s 121ms/step - loss: 19.2844 - mae: 19.7767
```

```
rnn_forecast = model_forecast(model, series[..., np.newaxis], window_size)
rnn_forecast = rnn_forecast[split_time - window_size:-1, -1, 0]
```

```
plt.figure(figsize=(10, 6))
plot_series(time_valid, x_valid)
plot_series(time_valid, rnn_forecast)
```



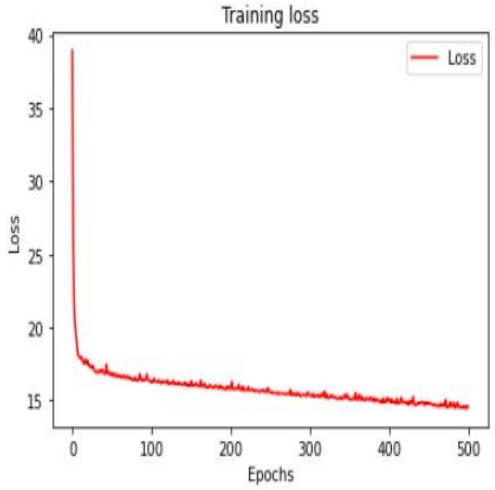

```
tf.keras.metrics.mean_absolute_error(x_valid, rnn_forecast).numpy()
```

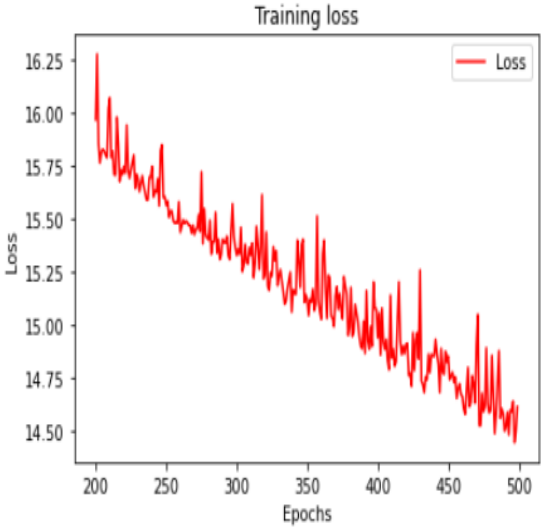
```
15.089574
```

```
import matplotlib.image  as mpimg
import matplotlib.pyplot as plt

#-----------------------------------------------------------
# Retrieve a list of list results on training and test data
# sets for each training epoch
#-----------------------------------------------------------
loss=history.history['loss']

epochs=range(len(loss)) # Get number of epochs


#------------------------------------------------
# Plot training and validation loss per epoch
#------------------------------------------------
plt.plot(epochs, loss, 'r')
plt.title('Training loss')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend(["Loss"])

plt.figure()
```

```
zoomed_loss = loss[200:]
zoomed_epochs = range(200,500)


#------------------------------------------------
# Plot training and validation loss per epoch
#------------------------------------------------
plt.plot(zoomed_epochs, zoomed_loss, 'r')
plt.title('Training loss')
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend(["Loss"])

plt.figure()
```

<Figure size 432x288 with 0 Axes>






```
print(rnn_forecast)
```

```
[135.92502    99.271904  105.50781   129.8834    109.807526  150.31667
 192.19649   168.62233   179.26598   126.94186   148.1779    169.16774
 158.67053   159.65434   195.03677   190.81548   185.02423   192.65051
 177.28357   201.87624   190.02647   177.29008   172.10686   182.73575
 159.40915   149.92252   153.19821   164.56311   159.92856   134.9303
 199.1364    128.0686    148.98816   218.37451   172.10358   183.2
 178.08499   152.958     152.39517   164.84302   174.30031   181.5857
 155.59616   163.76837   187.73816   164.62791   160.91759   153.94258
 147.61807   142.00415   104.43032   113.7124    108.309616   95.00554
 112.304596  115.40711   100.481155   90.23153    88.875725   77.811775
  71.621796   73.536644   77.760765   85.392105   72.36296    69.2541
  69.323746   74.63767    76.34568    55.404366   64.97453    76.79944
  50.71929    50.283485   46.622158   34.97485    36.70865    48.570408
  47.02358    52.555763   60.58049    34.326183   29.493862   27.580122
  43.976254   20.87406    17.527748   16.793663   14.4772415  14.254313
  14.442165   17.74445    18.561777   22.075977   21.341286   24.885897
  22.398823   23.092352   19.850086   13.912702   12.291941   11.018311
  13.696872   12.136144   12.551162   11.527196   10.368964    9.625668
```

.....

BIBLIOGRAPHY/REFRENCES

This project could not be possible without the help of online resources. Some of the resources includes:-

1. GitHub Resources and third party libraries.
2. Online videos & Tutorials.
3. Online QnA forums (Stackoverflow, quora etc.)
4. Babcok H.W., "The topology of the Sun's magnetic field and the 22-Year cycle", Astrophysical Journal, 2, pp. 572–587, 1961.
5. R. P. Kane, "An estimate for the size of sunspot cycle 24," *Solar Physics*, vol. 282, no. 1, pp. 87–90, 2013.View at: Publisher Site | Google Scholar
6. https://www.tensorflow.org/tutorials/structured_data/time_series
7. https://www.analyticsvidhya.com/blog/2018/02/time-series-forecasting-methods/
8. https://towardsdatascience.com/prediction-and-analysis-of-time-series-data-using-tensorflow-2136ef633018
9. https://www.ibm.com/cloud/learn/recurrent-neural-networks
10. https://www.analyticsvidhya.com/blog/2021/06/linear-regression-using-neural-networks/
11. https://machinelearningmastery.com/gentle-introduction-long-short-term-memory-networks-experts/

FUTURE WORK

We will try to add some extra functionality and apply some parameter tuning to the model and improve the accuracy.

We will also be testing with bidirectional LSTMS and see how much our accuracy improves.

Our goal is to get the **mean absolute error** value below 10.00.

THANK YOU