LSTM(LONG-SHORT TERM MEMORY) MULTI-STEP TIME SERIES FORECASTING USING MANY-TO-SEQUENCE PREDICTION.

Name : Krishna Pavan Kalyan Emani  
Course : *M.Sc Artificial Intelligence*  
Institution: Ulster UniversityAddress : United Kingdom  
Email Id : Emani-KPK@ulster.ac.uk

Abstract: LSTM is an array of gates utilized to overcome the vanishing gradient problem faced in RNN. Multi-Step time series forecasting is one of the techniques in LSTM algorithm used to compute time sequences of multiple inputs for forecasting future solar power for gas consumption in every individual house hold by implementing many-to-many sequence prediction. LSTM also consists of internal memory which acts as local variable in order to predict the next step collected from pervious input sequence. A distinct framework is designed in LSTM to make multi – time step predictions known as sequence-to-sequence model. An encoder understands the input sequences and flattens its memory into the local state and decoder learns input representations utilized to produce output sequence. RMSE and MAE are used as evaluation metrics to implement error forecast.

Keywords— RNN, LSTM, Many-To-Many, multi-Time step, Encoder, Decoder, RMSE, MAE.

# Introduction

Increasing of solar power for gas consumption in every day and solar power consumption design creates complicated task for network engineers to control the electrical power model.

RNN: Recurrent neural network (RNN) is high-level design consists of a set of locally concatenated with each other as hidden layers. RNN neural networks computes present input state of time ‘t’ learned from the prior input time step ‘t-1’. RNN memorizes the past time series to produce current output. it is mainly utilized to operate short-term sequences. Hidden layer stores the prior input used back at any step-in time sequence to determine the output. RNN reiterates the function more frequently with hidden layers using back-propagation method to measure erroneous difference between actual output and predicted output. (1)

The back-propagation method in RNN is implemented to correct the weights of input according to the expected output. the operational series models located in RNN useful to standardize input sequences into one to many and many to one. Back propagation has its own negative effects while computing in RNN. One of the significant errors is to recognize while balancing the weights of prior layers. The loss turns out to be insufficient to identify the faults in previous time sequences due to low recognition of memory stored in hidden layers is known as Vanishing gradient problem. (1)

input state

‘t’ output

‘t-1’ ‘t’ & ‘t-1’

FIG 1: INTRNAL RNN ARCHITECTURE OF RNN

## Where **‘**t’ is current input state

‘t-1’ is prior hidden input state

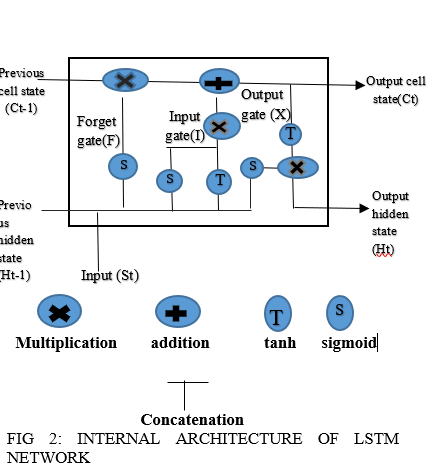
Output h(t) = t+t-1, tanh function sums both input hidden state,

Conventional RNN’s have drawback in processing long term sentences and texts and inefficient to filter unnecessary sequences. (1)

Long-short term memory (LSTM):

LSTM networks understands from RNN. LSTM is a method implemented in deep neural network for computing long term sequences effortless. LSTM network is capable to filter unwanted sequences and produce the output in very less time

Using forget gate and highly efficient in solving vanishing gradient problem. (1)



From the above figure, cell state shows present state of memory, and hidden state preserves un-essential and featured data, the forget gate eliminated all unnecessary sequences. this operation needs three gates a) forget gate b) input gate c) output gate. (1)

#### Forget gate: the primary stage of LSTM network is forgot gate, the summed vector inputs from previous hidden state(Ht-1) and input(St) can be transfer through sigmoid function (s) operates the vector inputs into two binary outputs ‘1’ shows to permit the information to proceed from forget gate and ‘0’ denotes to eliminate the data from forget gate. the mathematical equation for forget gate is given by. (1)

F=σ (Ht-1 & St)

#### Input gate: the series of both previous hidden state (Ht-1) and input (St) the two samples of vector input can be processed through sigmoid and activation function and the total multiplication of both the operations are shown below. (1)

I = σ (Ht-I & St) \* tanh (Ht-1 &St)

The computation for present cell output is derived from the input gate is given by:

Ct = (F \* Ct) + I

#### 

#### Output gate: the addition of two inputs from previous hidden state (Ht – 1) and input state (St) pssed through sigmoid function (St) and present cell transfers from input gate to tanh function produces the multiplication of computation as below:

*X = σ(Ht-1 & St) \* tanh(Ct)*

*TIME SERIES FORECASTING METHODS:*

Time series forecasting plays vital role in forecasting the future and it also called as predicting by examining the sequence from prior data. It is helpful in modeling results in choosen area. Information that is needed for time series forecasting consists od two methods: 1) time series data 2) data with time points. Time series data can be explained as. (3)

X =

Where X is time sequencs, xt is time complete, N is no of observations during that time, caluclation at an discrete time step is xt fot time step t

II) SINGLE-STEP FORECASTING:

Choosing of prediction class and techniques based on the needs of the designing path and reccurence of gathered data. Single-step is appropriate where short term prediction is essential. For instance, time span of many hours,days could all be reviewed as short term. For a task like that, caluclating a single time series ahead forecast is essential. Single step forecast (t+1) is done by trasfering the present and prior readings.(t,t-1,….,t-n) to a selected design..(3)

f(t+1) = m(o(t),o(t-1),o(t-2),…,o(t-n))

where f(t+1) is the prediction of time (t+1), m is the model and o(t) is reading at time t.(3)

III) MULTI-STEP FORECASTING

Direct step forecasting is essential when the path of design needs long-term time sequence forecasting. To caluaclate multiple step ahead forecasting requires direct H strategy method.(3)

Direct strategy: direct technique improves N individual prediction methods to predict N sequences , for instance initial forecast step (t+1) requires to be computed with a design and next with a unique design would be utilised to predict the second step by calucalting the example given below: (3)

f(t+1) = m1(o(t), o(t-1), 0(t-2),…..,o(t-n))

f(t+2) = m2(o(t), 0(t-1),o(t-2),…., o(t-n))

the direct multi-step technique can be equated as:

Yt+h = fh(Yt,….,Yt-n+1)

Wher h is the number of sequences to predict for the future, n is the autoregressive order of the design, fh is any aribitary intrepeter.

The direct technique doesn’t includes of other assembled faults due to it doesn’t need any predictes value as an input. It will not ensure any statistical reliance in predicting steps as every sequence is trained autonomously. (3)

IV) MODELS IN LSTM:

There are four types of models in lstm

1. One-to-one method
2. One-to-many method
3. Many-to-many method
4. Mayny-to-one method

ONE-TO-ONE METHOD:

A one-to-one method generates single output for every input value. The local state for the initial time point is zero,from the zero point, the local sate is gathered over previous time sequences.

In the scenario of sequence forecasting, this method generates ,single time point prediction for every observed time point acquired as input.

This is a inefficient utilization for RNNS as the design has no way to understand over input or output points. (3)

ONE-TO-MANY METHOD:

One-to-many model generates various output values for single input value.

The local state is gathered as separarte value in the output step is processed.

This method can be utilised for image heading when one image is furnished as input and a series of words are produced as output.

MANY-TO-ONE METHOD

A many-to-one design process single output value after accumulating various input values.

The local sate is gathered with separate input value prior a last output value is generated.

In the scenario of time sequence, this method would utilised as series of current readings to predict the further time point.this design would illustrate the classical autoregressive time sequence method.(3)

MANY-TO-MANY METHOD

A many-to-many method generates numerous outputs after accumulating various input values.

FIG 3:INTERNAL ARCHITECTURE OF MANY-TO-MANY SEQUENCE MODEL

As in the scenario of many-to-one, step is received until the initial output is generated, but various time points are output for this instance.

Mainly, the frequency of input time steps doesn’t need to map with the frequency of output time steps.

In the scenario of time sequence predicting, this method utilizes a series of current readings to construct a multistep prediction.

In a way, it concatenates the abilities of the many-to-one and one-to-many methods.(3)

SEQUENCE FORECASTING:

LSTM is mostly applicable for sequence information. LSTM can forecast, distribute, categorizes, and produce series information. A sequence is representing hierarchy of readings, more than a series of readings. For an instance sequence is a series of test when the timestamps and numerical data are in the hierarchy of the series.

Predictions depend on the series of data is known as sequence forecasting.(3)

Sequence forecasting is of four types.

1. Sequence numeric forecasting.
2. Sequence categorization
3. Sequence production
4. Sequence-to-sequence prediction.

SEQUENCE NUMERIC FORECASTING:

Sequence numeric forecasting is forecasting the upcoming data for input sequence. It is mainly utilized in stock market prediction and weather prediction. (3)

INPUT OUTPUT

FIG 4: LSTM SEQUENCE FORECASTING MODEL

SEQUENCE CATEGORIZATION:

Sequence categorization forecasts the class name for input series. its applications are fraud identification and categorization of students depend on their rankings. (3)

INPUT OUTPUT

FIG 5: LSTM SEQUENCE CATEGORIZATION MODEL

SEQUENCE PRODUCTION:

Sequence production is any time when we produce a unique output series then it exhibits same features of input series. It is mainly utilized in the applications like blogs, music generation. (3)

INPUT OUTPUT

FIG 6: LSTM SEQUENCE GENERATION MODEL

SEQUENCE-TO-SEQUENCE FORECASTING:

Sequence-to-sequence forecasting is when we calculate the upcoming series for a input series. Its mainly used in the applications of document classification and multi-step time series forecasting. (3)

FIG 7: LSTM SEQUENCE TO SEQUENCE MODEL

V) ENCODER-DECODER IMPLEMENTATION:

In RNN, LSTM every single input respected to an output for the similar time point. The encoder-decoder design for RNN was initiated to explain the sequence-to-sequence design. An encoder-decoder accumulates a given sequence and produces the most chances of further series as output.

The encoder is accountable for entering into the given time points and converting the whole sequence into a standard vector size known as context vector.

The decoder is accountable for passing into the output time steps while learning the context vector. (3)

U Decoder U U

w w w

V Encoder V V

FIG 8 : INTERNAL ARCHITECTURE OF ENCODER AND DECODER

Encoder:

The encoder is orderly arrangement of various recurrent steps of LSTM and RNN units. Every single unit receives one sample from input sequence, gathers data from that sample and passes ahead. (3)

The hidden layer vector H(t) is calculated with the function of selected step. The task implemented with the correct weights to the prior hidden state H(t-1) and the given vector X(t):

H(t) = f(wH(t-1) + VX(t))

The resulted hidden state vector H(t) includes every converted data from prior hidden presentations and past inputs. (3)

CONTEXT VECTOR:

The context vector is the resulted hidden state generated from the encoder segment of the design, and presents the first hidden state for the decoder. Its summarizes the data for every input sample s to support the decoder to build appropriate predictions.

DECODER:

The decoder includes in a order of various recurrent steps. Every single recurrent step receives a hidden state(t-1) from past step and generates output y^(t) and its individual hidden state S(t).

Hidden state S(t) is calculated in proportion with the function of the selected recurrent step:

S(t) = f(wH(t-1))

The output y^(t) is calculated with softmax function implementing the hidden state at the present time step S(t) concatenated with their appropriate weights, in a way to make a probability vector.

INTERNAL ARCHITECTURE FOR PROPOSED METHODOLOGY:

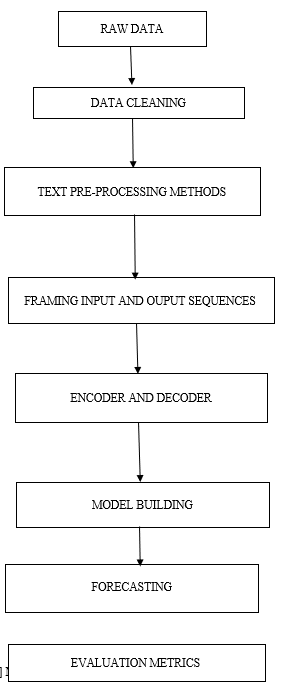


FIG 9 : INTERNAL ARCHITECTURE OF PROPOSED METHODOLOGY

VI) METHODOLOGY:

Import pandas text file using pd.read\_csv

import pandas as pd  
datase = pd.read\_csv('C:\\Users\\KALYAN\\Downloads\\pv.txt', header=0, low\_memory=False, infer\_datetime\_format=True, parse\_dates={'datetime':[0]}, index\_col=['datetime'])

import libraries required for data preprocessing

from numpy import nan  
from numpy import isnan  
from pandas import read\_csv  
from pandas import to\_numeric

fill the missed values with the same value of the old data given below:

def fill\_missing(values):  
 one\_da = 60 \* 24  
 for ro in range(values.shape[0]):  
 for colu in range(values.shape[1]):  
 if isnan(values[ro, colu]):  
 values[ro, colu] = values[ro - one\_da, colu]

Load the data set convert into numeric and give values to the dataset.

datase = read\_csv('C:\\Users\\KALYAN\\Downloads\\pv.txt', header=0, low\_memory=False, infer\_datetime\_format=True,  
 parse\_dates={'datetime': [0]}, index\_col=['datetime'])  
datase.replace('?', nan, inplace=True)  
datase = datase.astype('float32')  
fill\_missing(datase.values)  
values = datase.values  
datase.to\_csv('pv.csv')

convert the text file into .csv file using the following command and make datetime in a single column and add new column

from pandas import read\_csv  
datase = read\_csv('pv.csv', header=0, infer\_datetime\_format=True, parse\_dates=['datetime'], index\_col=['datetime'])  
daily\_group = datase.resample('D')  
daily\_dat = daily\_group.sum()  
print(daily\_dat.shape)  
print(daily\_dat.head())  
daily\_dat.to\_csv('pvr.csv')

import libraries required for data splitting, classification of input and output sequences, model building, multistep forecasting with many-to-many sequences using encoders and decoders.

from math import sqrt  
from numpy import split  
from numpy import array  
from pandas import read\_csv  
from sklearn.metrics import mean\_squared\_error  
from matplotlib import pyplot  
from keras.models import Sequential  
from keras.layers import Dense  
from keras.layers import Flatten  
from keras.layers import LSTM  
from keras.layers import RepeatVector  
from keras.layers import TimeDistributed

split the train and test dataset using the command below

def split\_datase(data):  
 trainx, testy = data[1:-328], data[-328:-6]  
 trainx = array(split(trainx, len(trainx) / 7))  
 testy = array(split(testy, len(testy) / 7))  
 return trainx, testy

Evaluate the forecasts of actual and prediction using the evaluation metrics RMSE,MSE and MAE

def evaluate\_forecast(actua, predicte):  
 score = list()  
 for i in range (actua.shape[1]):  
 mse = mean\_squared\_error(actua[:, i], predicte[:, i])  
 rmse = sqrt(mse)  
 score.append(rmse)  
 s = 0  
 for ro in range(actua.shape[0]):  
 for colu in range(actua.shape[1]):  
 s += (actua[ro, colu] - predicte[ro, colu]) \*\* 2  
 scores = sqrt(s / (actua.shape[0] \* actua.shape[1]))  
 return scores, score

summarizes the scores of the evaluation metrics using the following code

def summarize\_score(name, scores, score):  
 ss\_scores = ', '.join(['%.1f' % s for s in score])  
 print('%s: [%.3f] %s' % (name, scores, ss\_scores))

encoding and decoding the history into multiple inputs and outputs and step over the entire history one time step at a time and generate multistep time series for many-to-many sequences.

def too\_supervised(trainx, N\_input, N\_out=7):  
 data = trainx.reshape((trainx.shape[0] \* trainx.shape[1], trainx.shape[2]))  
 X, y = list(), list()  
 i\_start = 0  
 for \_ in range(len(data)):  
 i\_end = i\_start + N\_input  
 o\_end = i\_end + N\_out  
 if o\_end <= len(data):  
 X.append(data[i\_start:i\_end, :])  
 y.append(data[i\_end:o\_end, 0])  
 i\_start += 1  
 return array(X), array(y)

build the model with performance metrics like verbose, epochs, batch size, and the model sequential with two activation layers ‘relu’ and optimizer = ‘adam’

def build\_model(trainx, N\_input):  
 trainx\_x, trainy\_y = too\_supervised(trainx, N\_input)  
 verbose, epochs, batch\_size = 0, 50, 16  
 N\_timesteps, N\_features, N\_outputs = trainx\_x.shape[1], trainx\_x.shape[2], trainy\_y.shape[1]  
 trainy\_y = trainy\_y.reshape((trainy\_y.shape[0], trainy\_y.shape[1], 1))  
  
 model = Sequential()  
 model.add(LSTM(200, activation='relu', input\_shape=(N\_timesteps, N\_features)))  
 model.add(RepeatVector(N\_outputs))  
 model.add(LSTM(200, activation='relu', return\_sequences=True))  
 model.add(TimeDistributed(Dense(100, activation='relu')))  
 model.add(TimeDistributed(Dense(1)))  
 model.compile(loss='mse', optimizer='adam')  
 model.fit(trainx\_x, trainy\_y, epochs=epochs, batch\_size=batch\_size, verbose=verbose)  
 return model

build the forcast and flatten the data , accumulate all the previous observations for input data and forecast the upcoming week

def forecas(model, histor, N\_input):  
 data = array(histor)  
 data = data.reshape((data.shape[0] \* data.shape[1], data.shape[2]))  
 i\_x = data[-N\_input:, :]  
 i\_x = i\_x.reshape((1, i\_x.shape[0], i\_x.shape[1]))  
 yhat = model.predict(i\_x, verbose=0)  
 yhat = yhat[0]  
 return yhat

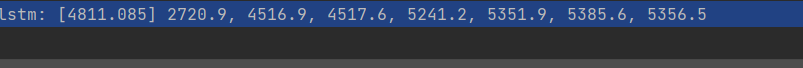
evaluate model and store the predictions in the history to make further predictions and evaluate prediction for each day in a week.

def evaluat\_model(trainx, testy, N\_input):  
 model = build\_model(trainx, N\_input)  
 histor = [x for x in trainx]  
 prediction = list()  
 for i in range(len(testy)):  
 yhat\_sequence = forecas(model, histor, N\_input)  
 prediction.append(yhat\_sequence)  
 histor.append(testy[i, :])  
  
 prediction = array(prediction)  
 scores, score = evaluate\_forecast(testy[:, :, 0], prediction)  
 return scores, score

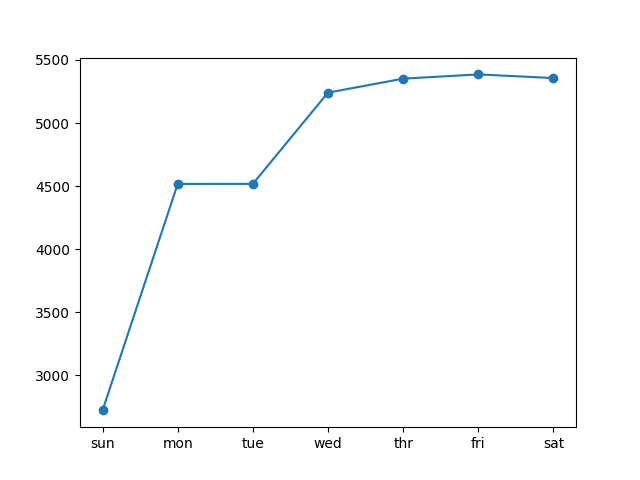
load the new dataset and select the no of inputs and summarize the score and plot the values.

datase = read\_csv('pvr.csv', header=0, infer\_datetime\_format=True, parse\_dates=['datetime'], index\_col=['datetime'])  
trainx, testy = split\_datase(datase.values)  
N\_input =14  
scores, score = evaluat\_model(trainx, testy, N\_input)  
summarize\_score('lstm', scores, score)  
days = ['sun', 'mon', 'tue', 'wed', 'thr', 'fri', 'sat']  
pyplot.plot(days, score, marker='o', label='lstm')  
pyplot.show()

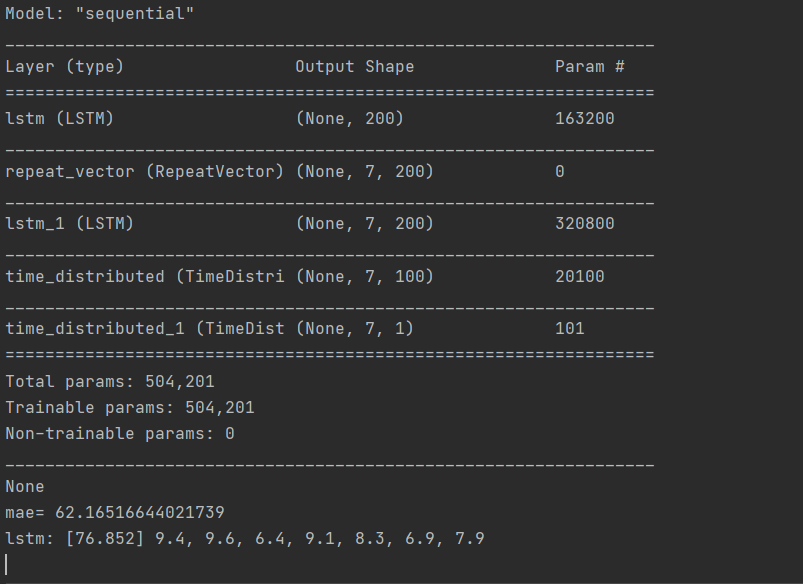
the predicted solar energy gas consumption for the next seven days.



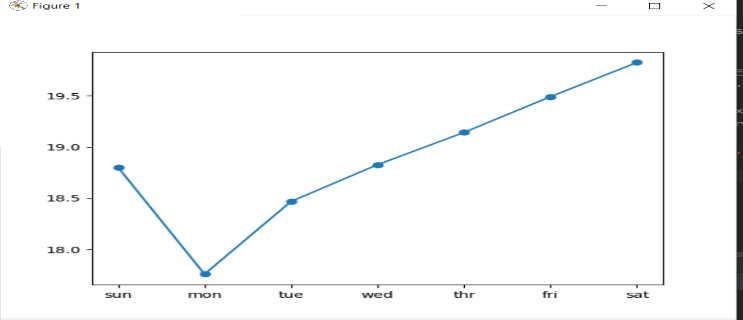
The RMSE,MSE AND MAE evaluation metrics for next 14 days.

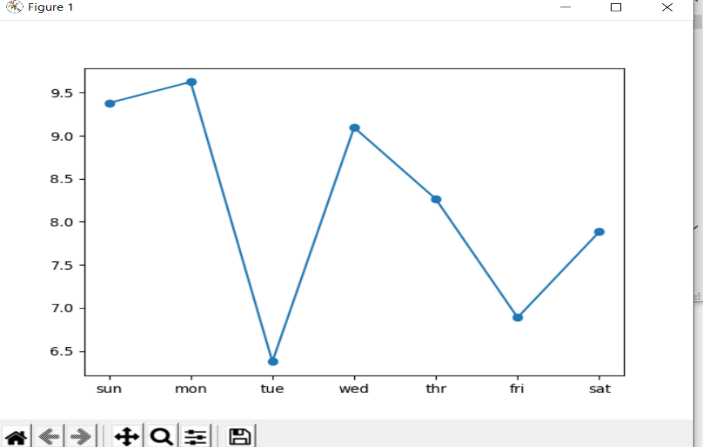


Performance evaluation metric and model summary with MAE

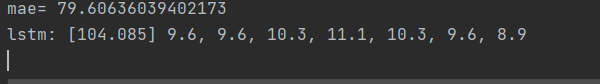


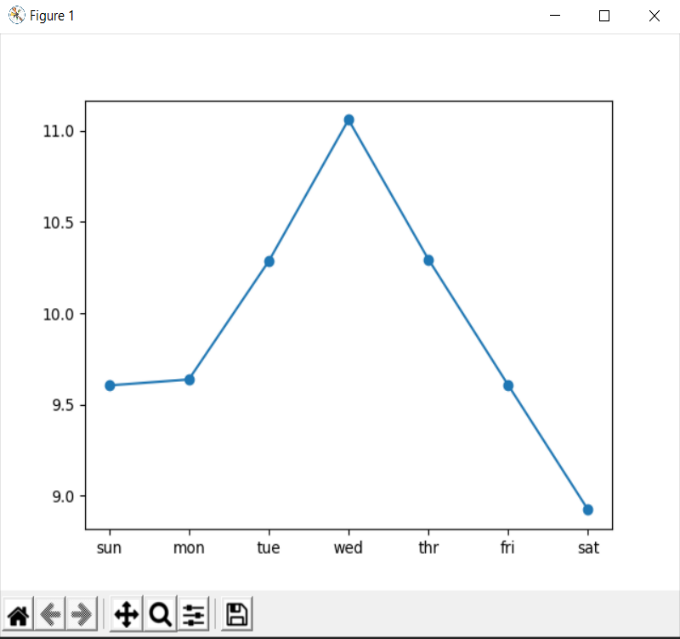
Evaluation metrics for 28 days





Evaluation metrics for 7 days





CONCLUSION:from the above evaluation metrics analysis we can clearly says that the aggregate solar power gas consumption for 7days,14 and 28 days are 76.85, 104.45, 410.80 respectively done exceptional.

REFERENCES:

### <https://learning.oreilly.com/library/view/neural-network-projects/9781789138900/5c3532cc-0b87-4c73-ab59-633172e048e1.xhtml>

2)[https://learning.oreilly.com/library/view/making-sense-of/9781430265931/A978-1-4302-6593- 1\_2\_Chapter.html](https://learning.oreilly.com/library/view/making-sense-of/9781430265931/A978-1-4302-6593-%20%201_2_Chapter.html)

3) https://learning.oreilly.com/library/view/deep-learning-with/9781484235164/html/456157\_1\_En\_9\_Chapter.xhtml