6203

ELECTRICITY PRICES PREDICTION

PHASE 1: Problem Definition and Design Thinking

INTRODUCTION

- In a Electricity Prediction, there are some Factors which affects the Prices which includes Electricity costs, demand and Supply Of the Electricity.
- Here , Long Short Term Memory (LSTM) is proposed which network is Capable of Learning Long Sequences with long time Lags.
- Many Exogenous Factors are also considered as inputs to the network , also includes the Forecasted System Demand , Historical Prices, Hour Of the day , Day of the week, Week of the year and holidays Information.
- We Propose To use LSTM model for 24Hours ahead Price Due to the strong Ability Of LSTM to memorize the previous price trend During Training.
- Simulation Models requires Detailed System Operation Parameters to build the model.
- They Focus More Quality Issues rather than during Learning Process .

METHODOLOGY

The Objective Of the Presented Methodology is to predict the day ahead Electricity Price, given historical Price data and exogenous Variables .

- A.)PREPROCESSING:

Both the negativeprices caused by Transmission Constraints and extremelyHigh prices caused by Shortage Of power Supply appear in two markets .

Logarithm Of the data To do the prediction Which is defined as: Ldt=ln(pdt)

d---→day

t----→time Step

P---→Electricity Price

- B.)LSTM NEUTRAL NETWORK:

The Weights update scheme may stop the Neutral Network from further training.

The key Idea Behind The LSTM is to regulate the cell States using Different Types of Gates includes Input, Forget and Output gates.

(Ct-1)----->State of the cell

Ct---------->Next state

ELECTRICITY PRICE FORECASTING USING LSTM

We Propose to apply Stacked LSTM with Multiple layers to Predict the electricity Price and this performance of the model can be influenced by the number of LSTM layers which includes Input time Steps, Structure of Forecasting Manner and Input variables.

The Actual Price Values at day d are denoted as:

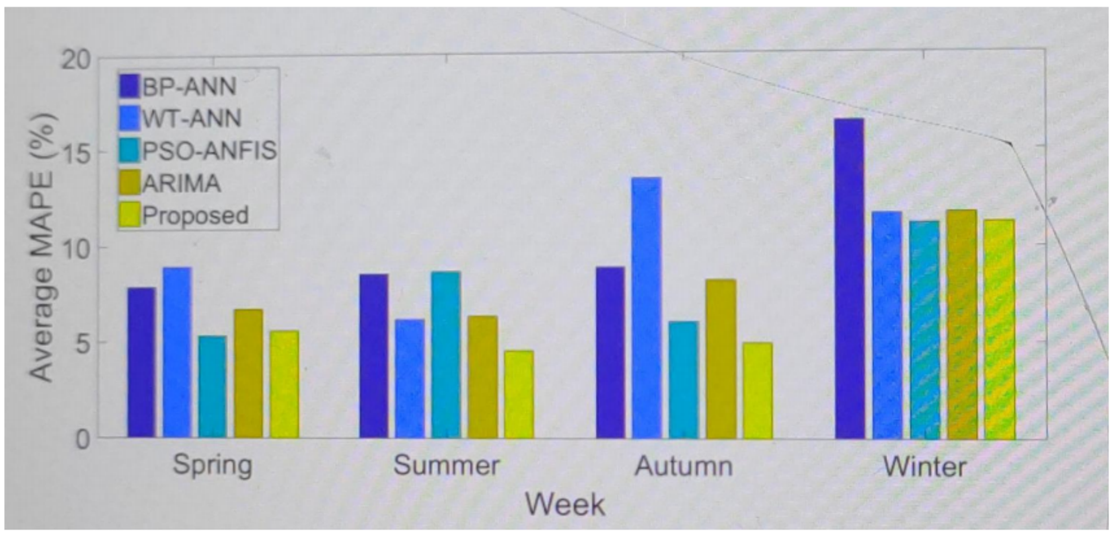
$$\hat{P}^d = \{\hat{p}_1^d, \hat{p}_2^d, \ldots, \hat{p}_t^d, \ldots, \hat{p}_T^d\}$$

The Structure of LSTM network based on a model in a recursive manner for multiple steps forecasting.

T can be 24 for a hourly market and 48 for a half hourly market.

CASE STUDY

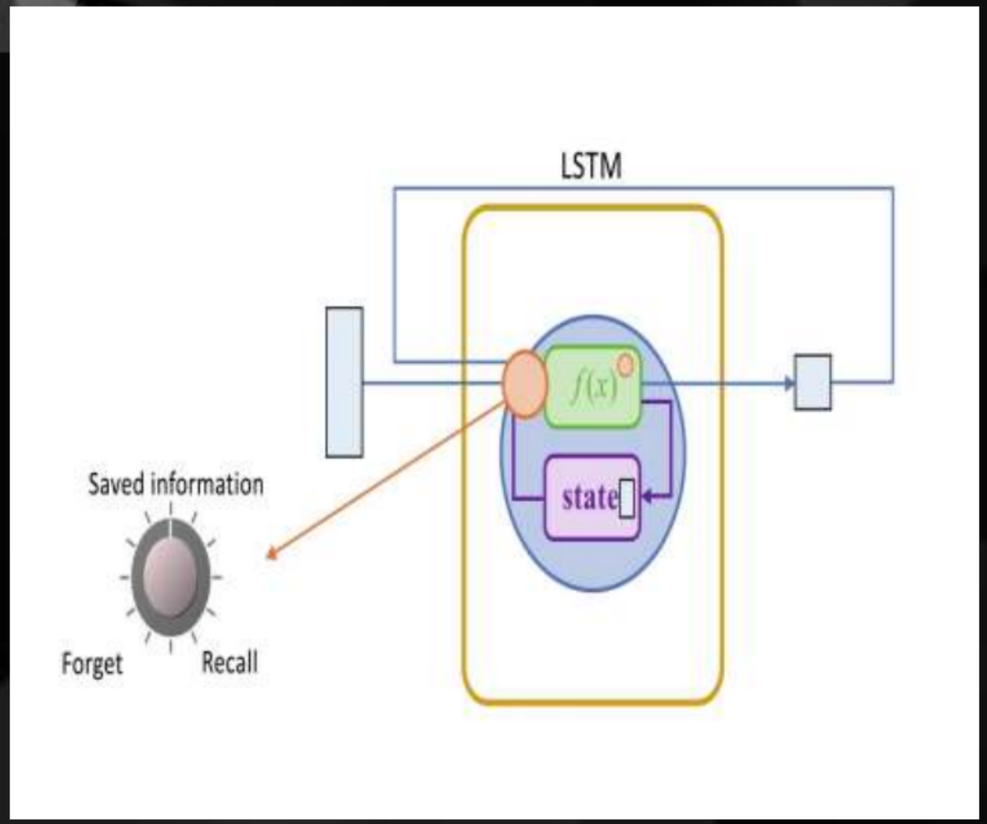
- The Price data and exogenous Variables from 3 months prior to the first day of each test week are used as training datasets and 20% of the training date is used for Validation.
- Four one-week periods representing four different seasons in 2013 are selected from VIC market for testing.



CONCLUSION

- We Proposed a Multilayer LSTM based model for forecasting the day-ahead Electricity Prices due to its ability to bridge long time Lags of inputs and remembering the historical trend Information in time Series.
- The Performance of the Proposed method is compared with other four popular methods used in the market (BP-ANN,WT-ANN,PSO-ANFIS and SARIMA)

Electricity Price Prediction Using LSTM



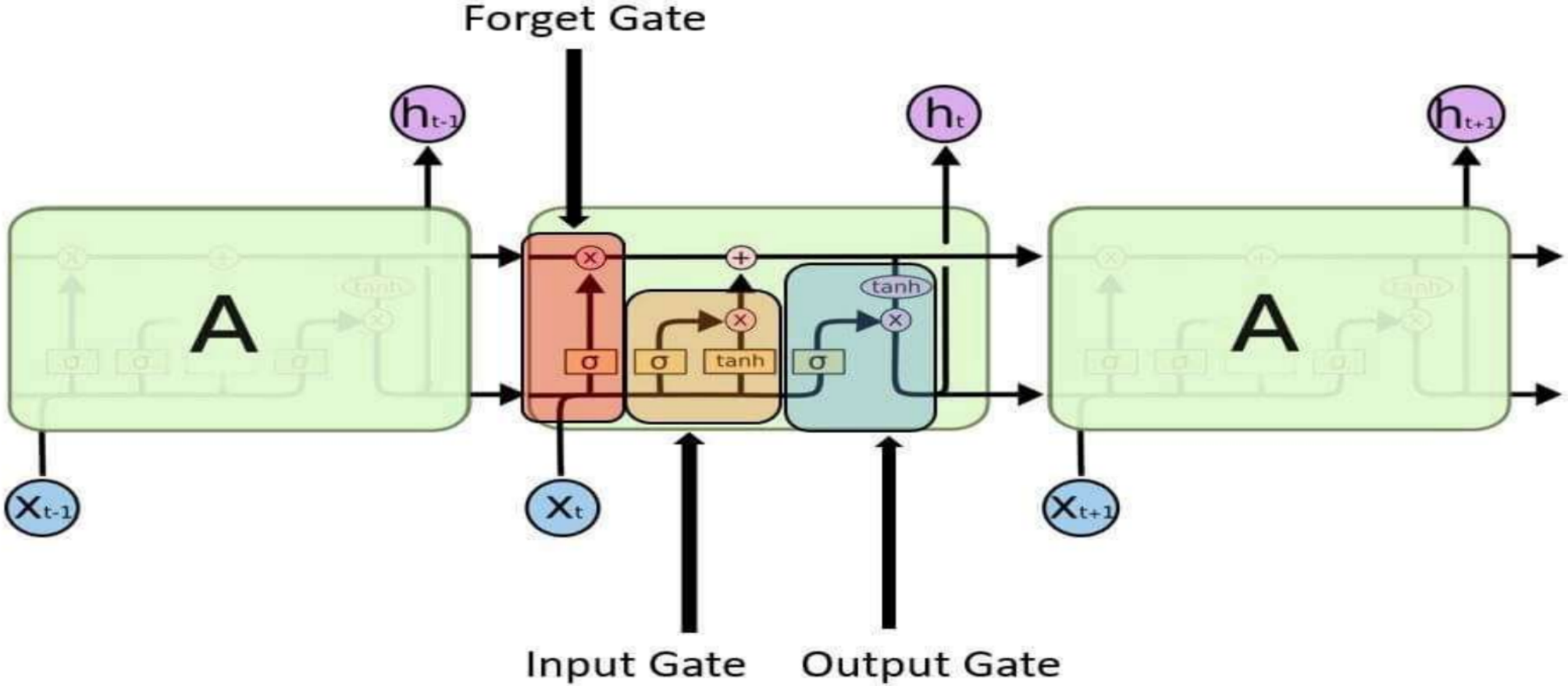
Contents

Introduction:

Long Short-Term Memory Networks is a deep learning, sequential neural network that allows information to persist. It can be used to predict electricity price

Steps involved:

How to solve LSTM Poblems:



Input gate:

It determines which of the input values should be used to change the memory. The sigmoid function determines whether to allow 0 or 1 values through. And the tanh function assigns weight to the data provided, determining their importance on a scale of -1 to 1.

Input Gate | Long Short Term Memory

$$i_t = \sigma(W_i . [h_t - 1, x_t] + bi)$$
$$C_t = \tanh(W_C . [h_t - 1, x_t] + b_C)$$

Forget Gate:

It finds the details that should be removed from the block. It is decided by a sigmoid function. For each number in the cell state Ct-1, it looks at the preceding state (ht-1) and the content input (Xt) and produces a number between 0 (omit this) and 1 (keep this).
Forget Gate | Long Short Term Memory

$$f_t = \sigma(W_f . [h_t - 1, x_t] + b_f)$$

Output Gate:

The block's input and memory are used to determine the output. The sigmoid function determines whether to allow 0 or 1 values through. And the tanh function determines which values are allowed to pass through 0, 1. And the tanh function assigns weight to the values provided, determining their relevance on a scale of -1 to 1 and multiplying it with the sigmoid output.

$$O_t = \sigma(W_o[h_t - 1, x_t] + b_o$$
$$h_t = o_t * \tanh(C_t)$$

Conclusion:

Long short-term memory (LSTM) is a deep learning architecture based on an artificial recurrent neural network (RNN). LSTMs are a viable answer for problems involving sequences and time series.

Data Science Project: LSTM

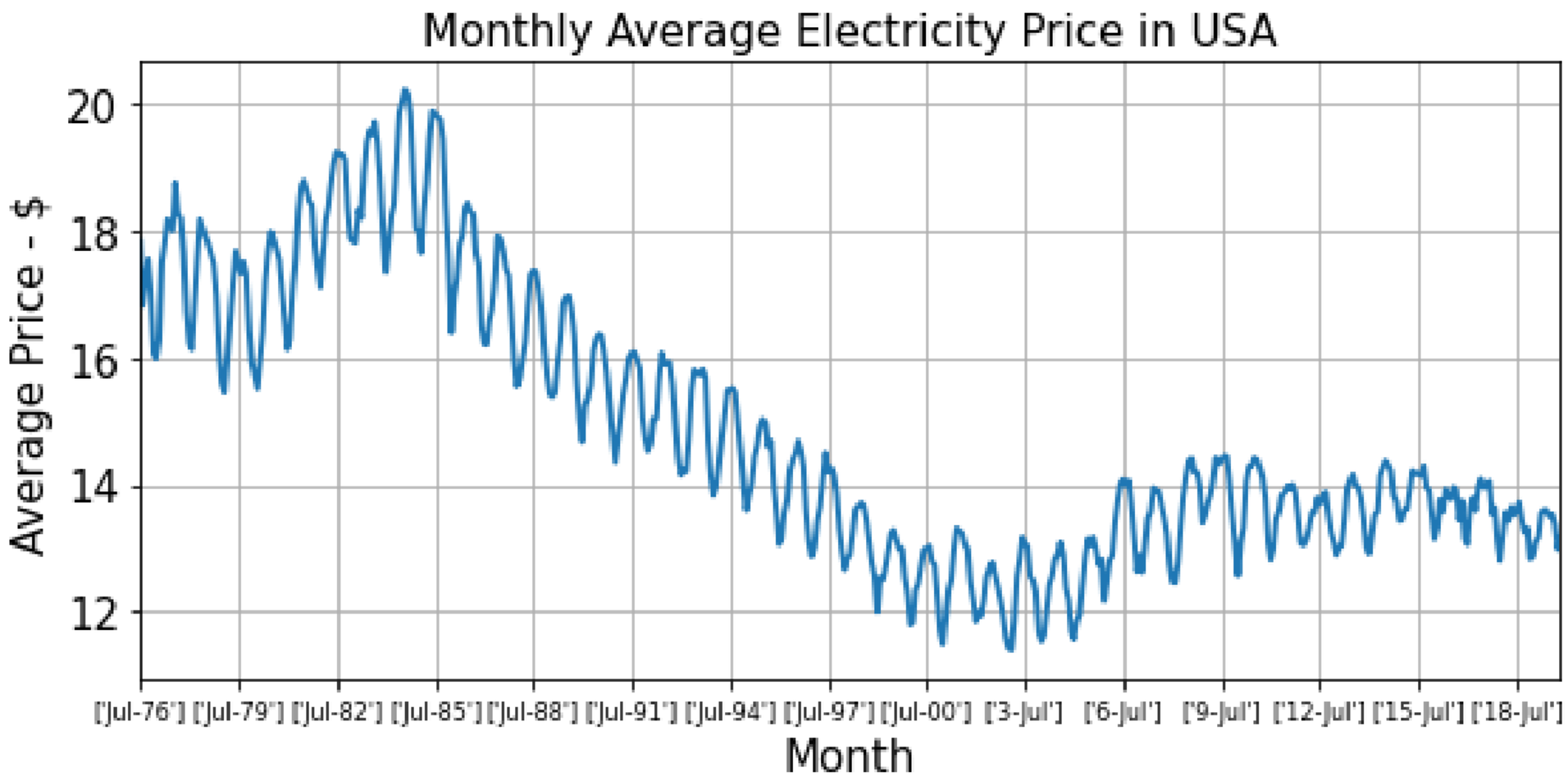
Setting up system

In [1]:
```
import numpy
import pandas
import matplotlib.pyplot as plt
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
```

In [2]:
```
# fix random seed for reproducibility
numpy.random.seed(7)
```

Loading and pre-processing data

In [7]:
```
# load the dataset
dataframe = pandas.read_csv('price.csv', usecols=[2], engine='python'
my_xticks = pandas.read_csv('price.csv', usecols=[1], engine='python'

dataset = dataframe.values
dataset = dataset.astype('float32')
my_xticks = my_xticks.values
my_xticks = my_xticks.astype('str')

#print(plt.rcParams.get('figure.figsize'))
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 9
fig_size[1] = 4
plt.rcParams["figure.figsize"] = fig_size

plt.plot(dataset,label='Real data',lw=2)

x = numpy.linspace(0,504,15)
x = x.astype(int)
my_xticks = [my_xticks[y] for y in x]
plt.xticks(x, my_xticks)

plt.xlabel('Month',fontsize=15)
plt.ylabel('Average Price - $',fontsize=15)
plt.title('Monthly Average Electricity Price in USA',fontsize=15)
plt.grid(b=None, which='major', axis='both')
plt.xlim([0,520])
plt.xticks(fontsize=8)
plt.yticks(fontsize=15)

plt.savefig('data.png', dpi=600)
plt.show()
```



In [4]:
```
# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
```

In [5]:
```
# split into train and test sets
train_size = int(len(dataset) * 0.794)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset
```

In [6]:
```
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return numpy.array(dataX), numpy.array(dataY)
```

In [7]:
```
# reshape into X=t and Y=t+1
look_back = 1
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
```

In [8]:
```
# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
```

Creating, training and testing the RNN/LSTM network

In [9]:

```
# create and fit the LSTM network
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
history=model.fit(trainX, trainY, epochs=10, batch_size=1, verbose=2)
```

```
Epoch 1/10
418/418 - 1s - loss: 0.1118
Epoch 2/10
418/418 - 1s - loss: 0.0345
Epoch 3/10
418/418 - 1s - loss: 0.0209
Epoch 4/10
```

```
418/418 - 1s - loss: 0.0098
Epoch 5/10
418/418 - 1s - loss: 0.0040
Epoch 6/10
418/418 - 1s - loss: 0.0022
Epoch 7/10
418/418 - 1s - loss: 0.0020
Epoch 8/10
418/418 - 1s - loss: 0.0021
Epoch 9/10
418/418 - 1s - loss: 0.0020
Epoch 10/10
418/418 - 1s - loss: 0.0020
```

In [10]:
```
# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
```

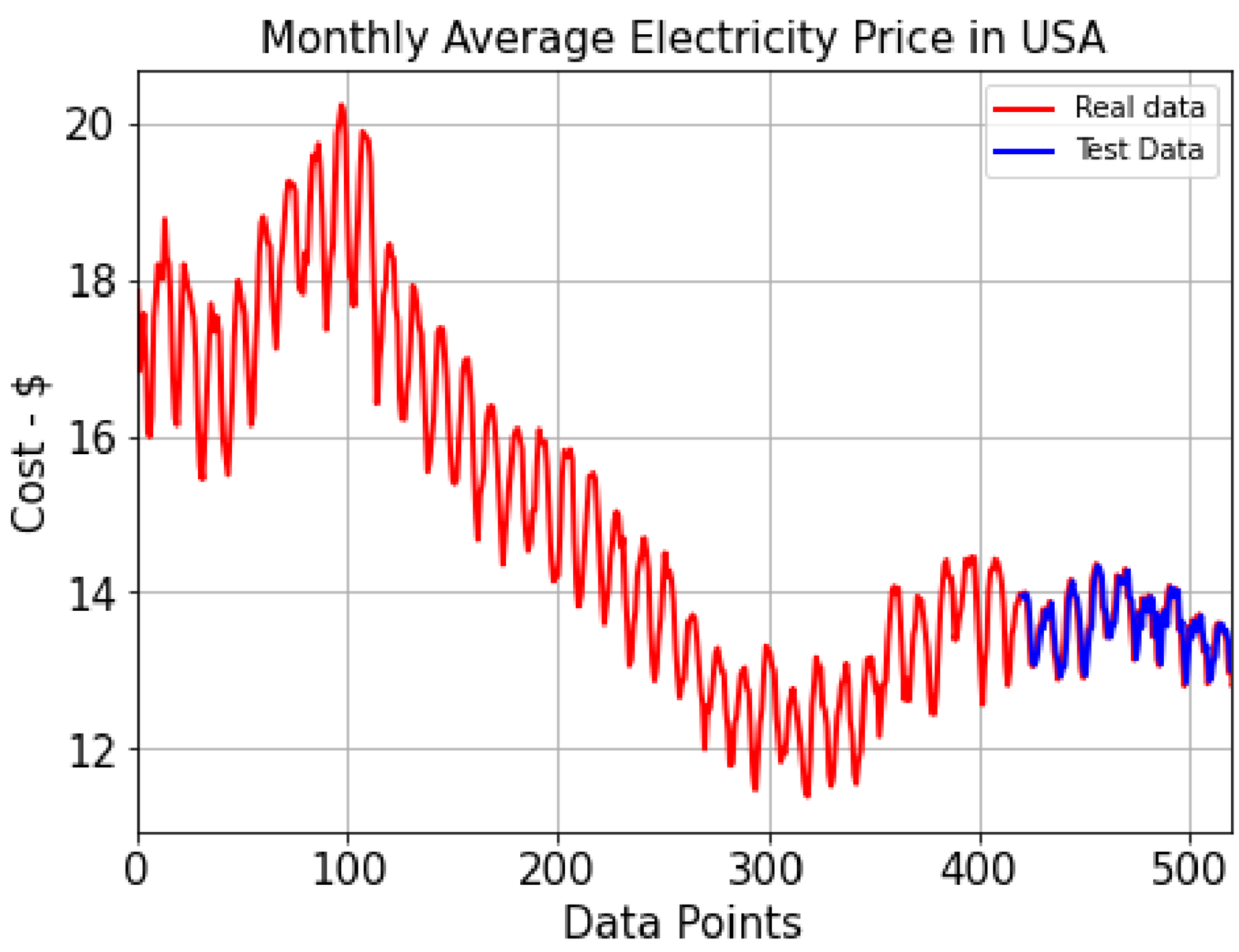
Results visualization

In [11]:
```
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])

testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])

# calculate root mean squared error
#trainScore = math.sqrt(mean_squared_error(trainY[0][0:-1], trainPred
#print('Train Score: %.4f RMSE' % (trainScore))
#testScore = math.sqrt(mean_squared_error(testY[0][0:-1],testPredict[
#print('Test Score: %.4f RMSE' % (testScore))

# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict))
print('Train Score: %.4f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0],testPredict))
print('Test Score: %.4f RMSE' % (testScore))
```

```
Train Score: 0.3936 RMSE
Test Score: 0.2496 RMSE
```

In [12]:
```
# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPre

# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :]

# plot baseline and predictions

#print(plt.rcParams.get('figure.figsize'))
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 7
fig_size[1] = 5
plt.rcParams["figure.figsize"] = fig_size

plt.plot(scaler.inverse_transform(dataset),label='Real data',color='r
#plt.plot(trainPredictPlot,label='Train Data',color='green')
```

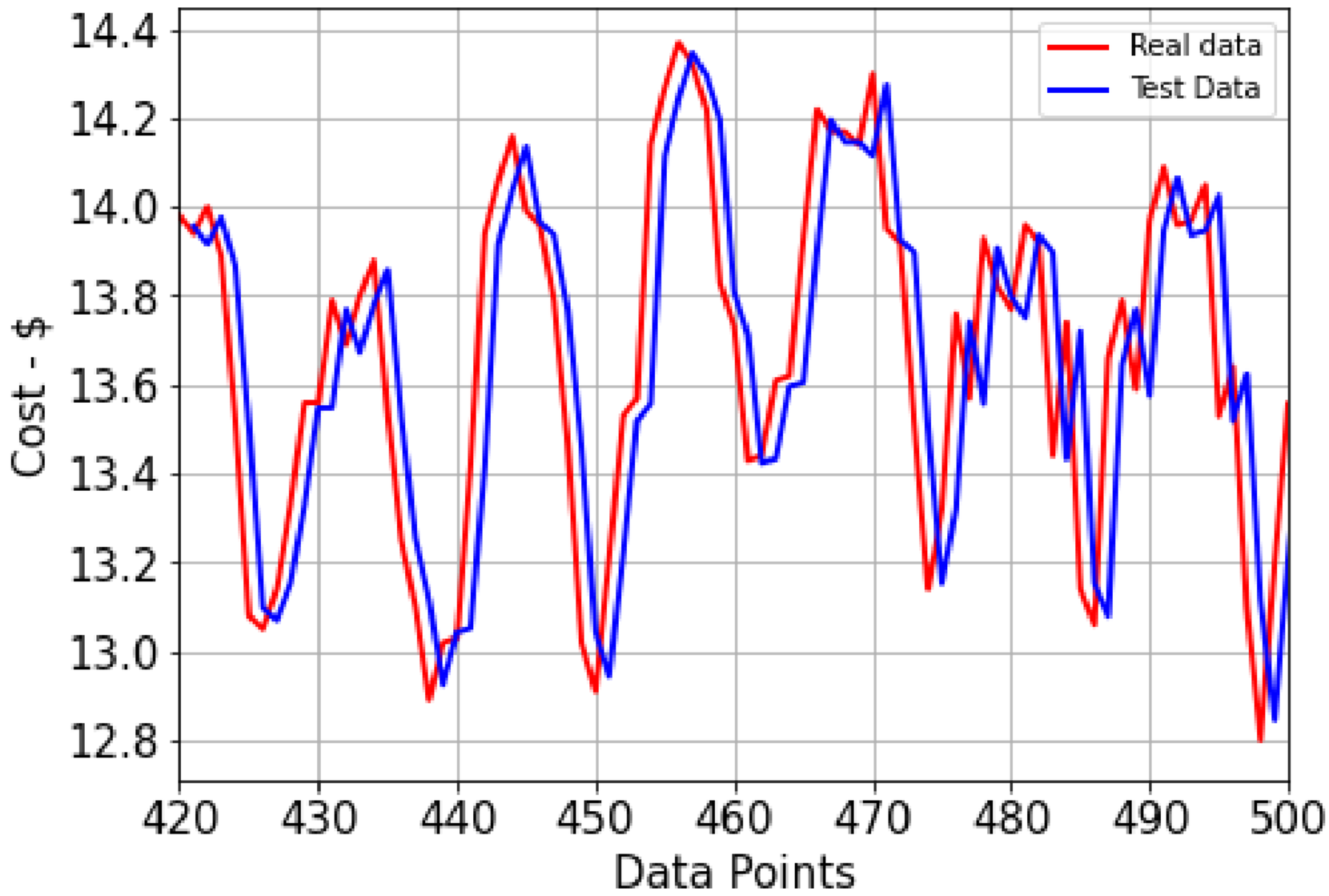
```
plt.plot(testPredictPlot,label='Test Data',color='blue',lw=2)

#plt.xticks(x, my_xticks)

plt.xlabel('Data Points',fontsize =15)
plt.ylabel('Cost - $',fontsize =15)
plt.title('Monthly Average Electricity Price in USA',fontsize =15)
plt.grid(b=None, which='major', axis='both')
plt.legend()
plt.xlim([0,520])
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

plt.savefig('LSTM.png', dpi=600)
plt.show()
```



In [16]:

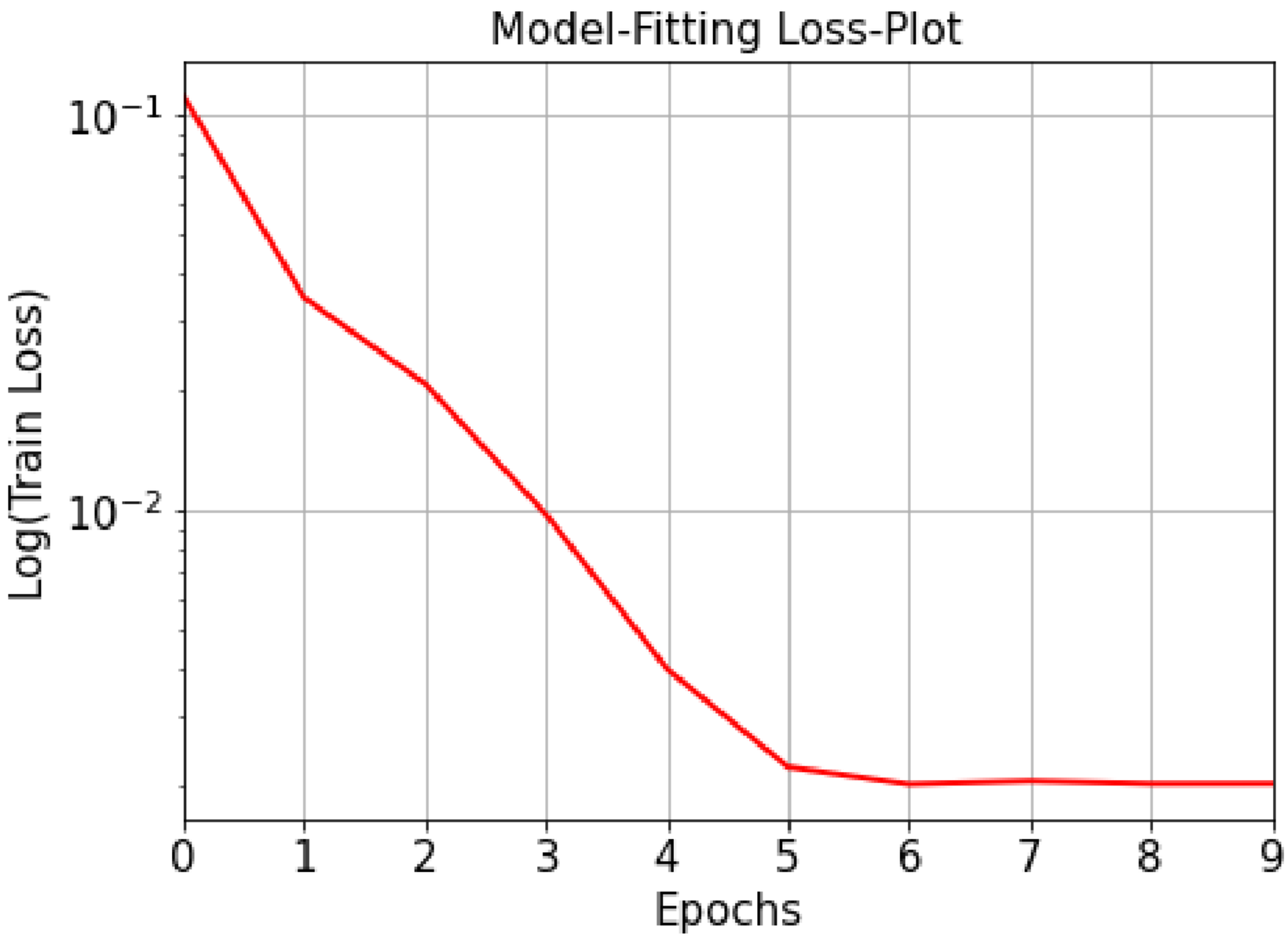
```
# plot baseline and predictions

#print(plt.rcParams.get('figure.figsize'))
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 7
fig_size[1] = 5
plt.rcParams["figure.figsize"] = fig_size

OriginalPlot = scaler.inverse_transform(dataset)

OriginalPlot_new = OriginalPlot[420:525,:]
trainPredictPlot_new = trainPredictPlot[420:525,:]
testPredictPlot_new = testPredictPlot[420:525,:]

plt.plot(range(420,525),OriginalPlot_new,label='Real data',color='red
#plt.plot(trainPredictPlot_new,label='Train Data')
plt.plot(range(420,525),testPredictPlot_new,label='Test Data',color='

#plt.xticks(x, my_xticks)

plt.xlabel('Data Points',fontsize =15)
plt.ylabel('Cost - $',fontsize =15)
plt.title('Monthly Average Electricity Price in USA',fontsize =15)
plt.grid(b=None, which='major', axis='both')
plt.legend()
```

```
plt.xlim([420,500])
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

plt.savefig('LSTM_Zoom.png', dpi=600)
plt.show()
```



In [14]:

```
labels = ["loss"]
for lab in labels:
    plt.plot(history.history[lab],color='red',lw=2)

#print(plt.rcParams.get('figure.figsize'))
fig_size = plt.rcParams["figure.figsize"]
fig_size[0] = 7
fig_size[1] = 5
plt.rcParams["figure.figsize"] = fig_size

plt.yscale("log")
plt.xlabel('Epochs',fontsize =15)
plt.ylabel('Log(Train Loss)',fontsize =15)
plt.title('Model-Fitting Loss-Plot',fontsize =15)
plt.grid(b=None, which='major', axis='both')

plt.xlim([0,9])
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)

plt.savefig('Loss.png', dpi=600)
plt.show()
```



In [15]:
```
import csv
with open('data.csv', 'w',newline='') as csvfile:
    # creating a csv writer object
    csvwriter = csv.writer(csvfile)

    # writing the data rows
    csvwriter.writerows(testPredictPlot)
```

In [ ]: