Deep Learning for hourly load forecasting of a residential zonal area in Thane, Mumbai

A Project Report
Submitted in partial fulfillment of
the requirements for course project of
Advanced Machine Learning (CS726)
by

Bharat Bohara
(Roll No. 173172001)

Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Mumbai 400076 (India)

1 May 2018

Acceptance Certificate

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay

The course project report entitled “Deep Learning for hourly load forecasting of a residential zonal area in Thane, Mumbai” submitted by Bharat Bohara (Roll No. 173172001) may be accepted for being evaluated.

Date: 1 May 2018

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I declare that I have properly and accurately acknowledged all sources used in the production of this report. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Bharat Bohara

Date: 1 May 2018 (Roll No. 173172001)

Abstract

More than 60 –70 % of energy is consumed daily on heating and cooling load in residential as well as industrial sectors. So efficient energy consumption and sustainability via minimizing excessive energy waste is the prior need of these days. It can be done by intelligent distribution as well as consumption of energy at the end usage level. But for the intelligent decision making of how to use the energy primarily requires accurate prediction of the future energy loads and demands. Future energy load forecasting has acquired lot of attention in the recent past, however it has been found to be difficult problem. Previously due to lack of higher computational power, the conventional methods of foresting were not powerful enough to fill up the gap of the mismatch between original and predicted forecasts. However, these days along with the higher computation achievement and innovations in new forecasting methodology based on deep neural networks, specially Long Short Term Memory (LSTM) algorithms have proved to be the promising techniques in predicting the future outcomes more accurately as compared to that of the conventional methods.

In this course project, 90 % of the historical data of power consumption in one of the location in Thane, Mumbai has been taken to train LSTM based sequence to sequence (S2S) neural networks architecture, while the remaining 10 % of the fetched data has been used for the validation of the designed deep neural network based model. For faster computation and con-sizing the whole project, Keras library along with Tensorflow as blackened has been used. Finally, the trained model is further used to predict the future electric load demands. As the continuation of this project, the model can be modified to adaptive predictive model for online load forecasting; that will be more accurate by continuously fetching data online and training the model. The predicted results of the model are depicted along with this report.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Buildings are the major energy consumer worldwide, accounting for 40 % 60 % of the total energy production for heating, cooling and minor in lighting and other purposes. In addition to being a major energy consumer, buildings are shown to account for a significant portion of energy wastage as well. As energy wastage poses a threat to sustainability, making buildings energy efficient is extremely crucial. Therefore, in making building energy consumption more efficient, it is necessary to have accurate predictions of its future energy consumption.

At the grid level, to minimize the energy wastage and making the power generation and distribution more efficient, the future of the power grid is moving to a new paradigm of smart grids. Smart grids are promising, unprecedented flexibility in energy generation and distribution. In order to provide that flexibility, the power grid has to be able to dynamically adapt to the changes in demand and efficiently distribute the generated energy from the various sources such as renewable energy. Therefore, intelligent control decisions should be made continuously at aggregate level as well as modular level in the grid. In achieving that goal and ensuring the reliability of the grid, the ability of forecasting the future demands is important.

Further, demand or load forecasting is crucial for mitigating uncertainties of the future. In that, individual building level demand forecasting is crucial as well as forecasting aggregate loads. The advent of smart meters have made the acquisition of energy consumption data at building and individual site level feasible. Thus data driven and statistical forecasting models are made possible.

Aggregate level and building level load forecasting can be viewed in three different categories: 1) Short-term 2) Medium- term and 3) Long-term. It has been determined that the load forecasting is a hard problem and in that, individual building level load forecasting is even harder than aggregate load forecasting. Mainly there are two methods for performing energy load forecasting: 1) Physics principles based models (White âĂŞ box models) and 2) Statistical and machine learning based models (Data driven or black âĂŞ box models). In this course project, second category of forecasting using deep learning has been used and presented in this report. Despite the extensive research carried out in this area, individual site level load forecasting remains to be a difficult problem.

Therefore, as per this course project, deep learning based methodology for performing individual building level load forecasting has been implemented using some open source libraries i.e. Keras along with Tensorflow as an blackened. Deep learning allows models composed of multiple layers to learn representations in data. The use of multiple layers allow the learning process to be carried out with multiple layers of abstraction.

Chapter 2

Literature Review

LONG SHORT TERM MEMORY (LSTM)

Recurrent Neural Networks (RNN) are usually trained using either Back-propagation through time or Real Time Recurrent Learning algorithm. Training with these methods often fails because of vanishing/exploding gradient. LSTM is a recurrent neural network that was specifically designed to overcome the problems of vanishing gradient, providing a model that is able to store information for long periods of time. (Goodfellow *et al.* (2016))

An LSTM network is comprised of memory cells with self bouncing loops. The self loop allows it to store temporal information encoded on the cell state. The flow of information through the network is handled by writing, erasing and reading from the cell memory state. These operations are handled by three gates respectively: 1) input gate, 2) forget gate and 3) output gate. (Goodfellow *et al.* (2016))

$$i_g = sigmoid(i[t] * W * i_x + o[t-1] * W * i_m + b_i) \tag{2.1}$$

$$f_g = sigmoid(i[t] * W * f_x + o[t1] * W * f_m + b_f) \tag{2.2}$$

$$o_g = sigmoid(i[t] * W * o_x + o[t1] * W * o_m + b_o) \tag{2.3}$$

$$u = tanh(i[t] * W * u_x + o[t1] * W * u_m + b_u) \tag{2.4}$$

$$x[t] = f * g * *x[t1] + i_g * u \tag{2.5}$$

$$o[t] = o * g * tanh(u) \tag{2.6}$$

where, ig corresponds to the input gate, fg to the forget gate and og to the output gate. Similarly, x[t] is the value of the state at time step t, o[t] is the the output of the cell and u is the update signal.

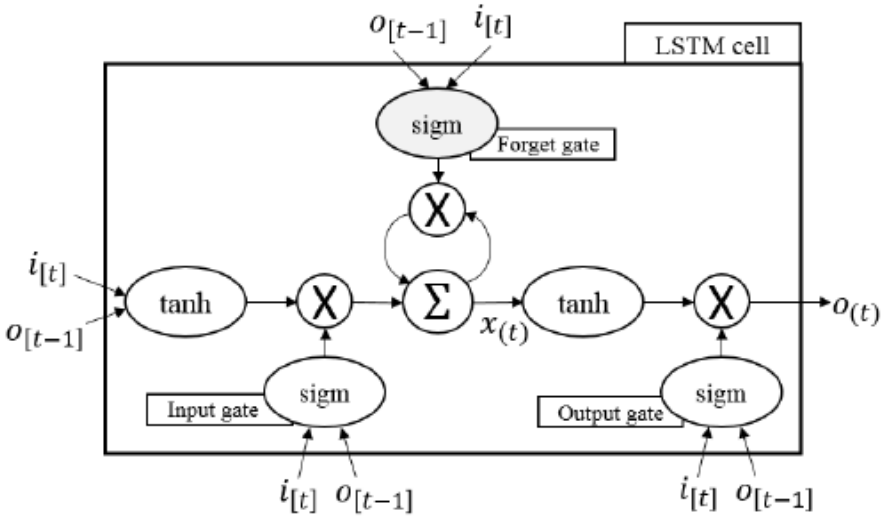

Figure 2.1: LSTM Cell (Marino *et al.* (2016))

The input gate decides if the update signal should modify the memory state or not. This is done by using a sigmoid function as a soft switch, whose on/off state depends on the current input and previous output. If the value of the input gate (ig) is close to zero, the update signal is multiplied by zero, therefore the state will not be affected by the update. Forget and output gates work in a similar manner.

LOAD FORECASTING USING DEEP NEURAL NETWORKS

This methodology is based on the load forecasting using sequence-to-sequence LSTM architecture. The objective of the presented methodology is to accurately estimate the electricity load (active power) for a time step or multiple time steps in the future, given historical electricity load data i.e. having load measurements available.

To train the model, back-propagation through time is used by the network that is unrolled by a fixed number of time steps in order to train the network using a gradient based method such as Stochastic Gradient Descent (SGD). The objective function that is minimized can be expressed as:

For training, ADAM optimization algorithm is used as the gradient based optimizer, instead of SGD. ADAM outperformed SGD in terms of faster convergence and lower error ratios. The unrolling was implemented with 23 steps.

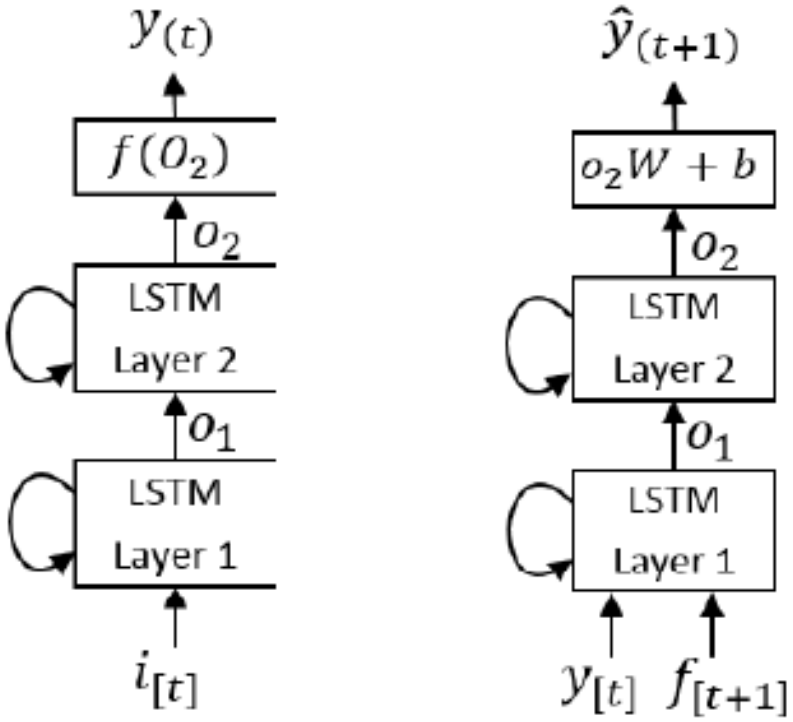

Figure 2.2: LSTM Architecture used for Load Forecasting. (Marino *et al.* (2016))

In order to further improve the flexibility of the load forecasting methodology, a different architecture based on LSTM, called sequence to sequence (S2S), can be implemented. S2S is an architecture that was proposed to map sequences of different lengths. The S2S architecture that is employed for load forecasting consists of two LSTM networks: encoder and a decoder. The task of the encoder is to convert input sequences of variable length and encode them in a fixed length vector, which is then used as the input state for the decoder. Then, the decoder generates an output sequence of length n. In this instance, that output sequence is the energy load forecast for the next n steps.

The main advantage of this architecture is that it allows inputs of arbitrary length. i.e. an arbitrary number of available load measurements of previous time steps can be used as inputs, to predict the load for an arbitrary number of future time steps. To perform the prediction, the outcome of the previous layer is used as the input for the encoder together with the corresponding date and time for and time is used as input. For training,

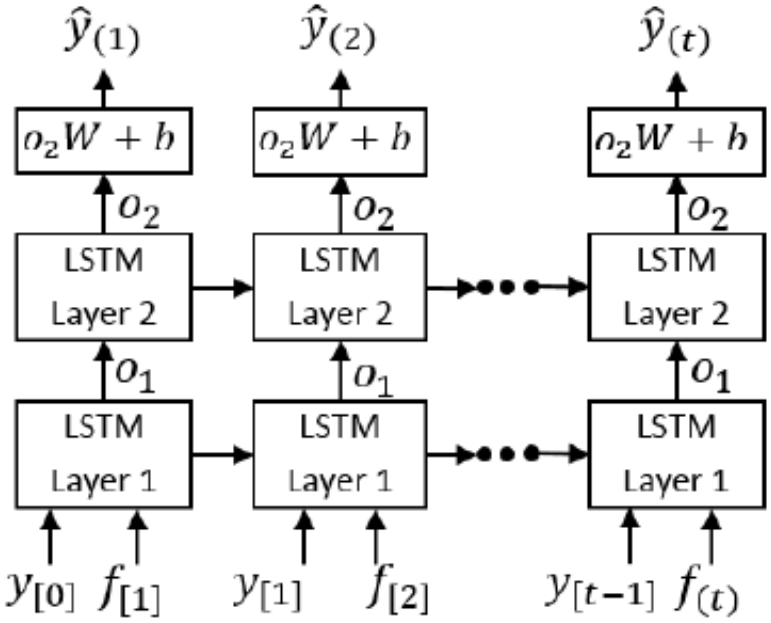

Figure 2.3: LSTM network used for the future load predictions.(Marino *et al.* (2016))

the encoder network is pre-trained to minimize the following error.

The back-propagation signals are allowed to flow from the decoder to the encoder. Therefore, weights for both encoder and decoder are updated in order to minimize the objective function. Both decoder and encoder are updated because the pre-training of the encoder alone is insufficient to achieve good performance. (Goodfellow *et al.* (2016))

Chapter 3

Datasets and Results

The presented methods was implemented on a dataset of electricity consumption for an aggregated residential and industrial load of entire ocation of Thane circle, Bhandup zonal region . The data set contained power consumption measurements gathered between July 2017 to December 2017 with one-hour resolution. The dataset contained aggregate active power load for the whole substation No/ Name: 113015/Lokpriya, Feeder No/ Name: 202/Udayshree feeder KV- 11/Outgoing recored by Maharashtra state electricity distribution Co. Ltd. In this project, the entire aggregate active load values for the whole substation feeder on hourly basis has been used due to difficulty in getting individual household data for every hour.

The LSTM architecture was tested on one-hour resolution simply bypassing the input from the current step straight to the output, this is because consequent measurements are very similar. Therefore, if the network predicts that the load for the next time step is the same that the load on the current time. Thus, the neural network is learning a naive mapping, where it generates an output equal to the input.

The proposed architecture is able to produce very low errors on training dataset by increasing the capacity of the network by increasing the number of layers and units. However, increasing the capacity of the network does not guarantee in improving the performance on testing data. In order to improve accuracy on testing data dropout (=0.2) has been used used as regularization methodology.

The load predictions model output on the test data on 90 % of the entire data set is found to be as shown below.

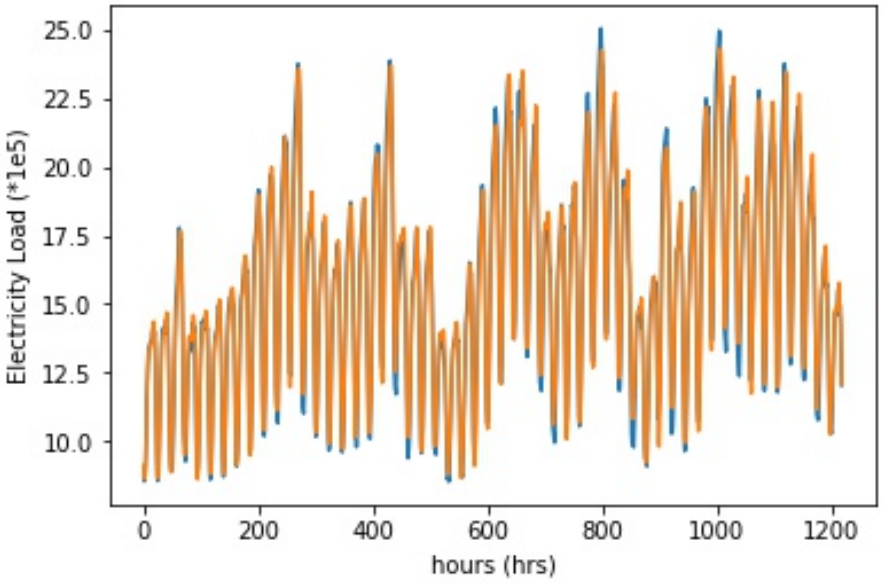

Figure 3.1: LSTM network used for the future load predictions.

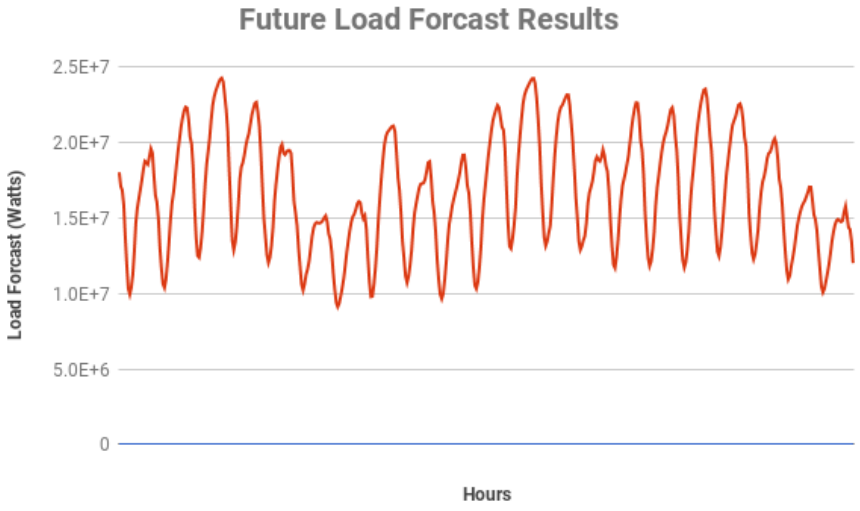

Figure 3.2: Future Forecast Results obtained from the Trained LSTM Model.

Chapter 4

Conclusion

The goal of this course project was to observe the effectiveness in using LSTM based deep neural networks for building level energy load forecasting. However, to compare the effectiveness of these algorithms, each algorithms need to be tested on different real world datasets as future work. Further, regularization approaches was applied to improve the generalization of the models. For the easy computation and scrutanizing the whole project predefined open source available libraries like Keras along with Tensorflow were applied. The results of the trained model was very close the original values which showd that deep learning based LSTM algorithms can be used for forcasting any time series datasets. From this project, conceptual insight behind the deep learning especially sequential recurrent network (LSTM, GRU etc) were learnt.

References

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y., 2016, *Deep learning*, Vol. 1 (MIT press Cambridge).

Marino, D. L., Amarasinghe, K., and Manic, M., 2016, "Building energy load forecasting using deep neural networks," in *Industrial Electronics Society, IECON 2016-42nd Annual Conference of the IEEE* (IEEE). pp. 7046–7051.

Appendix A

Course Project Source Code

Deep Learning for hourly load forecasting of the building in Thane, Mumbai

In [1]:

```
from __future__ import print_function
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM
from keras.models import Sequential
```

```
/home/varat/anaconda3/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning: Conversion of the second argument of issubdtype fro
m `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

In [10]:

```
# Load raw data
df = pd.read_csv('hourly_load_mumbai.csv', header=None)
# numpy array
df_array = df.values
# Daily load
daily_load = [df_array[i,:] for i in range(0, len(df)) if i%24 == 0] # list_daily_load changed to daily_load
# hourly load (23 hours for each day)
hourly_load = [df_array[i,1]/100000 for i in range(0, len(df)) if i%24 != 0] # list_hourly_load changead as above
# the length of the sequence for predicting the future value
seq_len = 23
load = []   # matrix changed simply to load

for i in range(len(hourly_load) - seq_len + 1):
      load.append(hourly_load[i:i + seq_len])
final_load = load     # matrix_load changed to final_load

# Shift all data by mean
final_load = np.array(final_load)
load_mean = final_load.mean()  # Shifted_value changed to load_mean
final_load -= load_mean       # This is after removing all the means from the original data
print('Data shape:', final_load.shape)
df_array
```

```
Data shape: (4877, 23)
```

Out[10]:

```
array([['1/1/2016', 1311693.43],
       ['1/1/2016 1:00', 1240583.19],
       ['1/1/2016 2:00', 1185502.49],
       ...,
       ['7/31/2016 21:00', 1427712.44],
       ['7/31/2016 22:00', 1315301.39],
       ['7/31/2016 23:00', 1204251.82]], dtype=object)
```

Split dataset: 90% for training and remaining 10% for testing

In [11]:

```
train_row = int(round(0.75 * final_load.shape[0]))
train_set = final_load[:train_row,:]

# Shuffle the training set (but do not shuffle the rest set)
np.random.shuffle(train_set)

# The training Set
X_train = train_set[:,:-1]
# The last column is the true value to compute the mean-squared-error loss
Y_train = train_set[:,-1]

# The test set
X_test = final_load[train_row:, :-1]
Y_test = final_load[train_row:, -1]
```

In [12]:

```
# Input to LSTM layer needs to have the shape of (number of samples, dimension of each element)
# Arranging the dimensions of inputs in order to feed into LSTM networks
```

```
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
```

Build the model

In [13]:

```
# Model Definition
model = Sequential()

# Layer 1: LSTM
model.add(LSTM(input_dim =1, output_dim = 50, return_sequences=True)) #activation by default=tanh
model.add(Dropout(0.2))

# Layer 2: LSTM
model.add(LSTM(output_dim=100, return_sequences=False))
model.add(Dropout(0.2))

# Layer 3: Dense
model.add(Dense(output_dim=1, activation='linear'))

#Compile the model
model.compile(loss='mse', optimizer='adam' )  # optimizer='rmsprop'
```

```
/home/varat/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:5: UserWarning: The `input_dim` and `input_length` arguments in recurrent layers are deprecated. Use `input_shape` instead.
  """
/home/varat/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:5: UserWarning: Update your `LSTM` call to the Keras 2 API: `LSTM(return_sequences=True, input_shape=(None, 1), units=50)`
  """
/home/varat/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:9: UserWarning: Update your `LSTM` call to the Keras 2 API: `LSTM(return_sequences=False, units=100)`
  if __name__ == '__main__':
/home/varat/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:13: UserWarning: Update your `Dense` call to the Keras 2 API: `Dense(activation="linear", units=1)`
  del sys.path[0]
```

Train the model

In [14]:

```
model.fit(X_train, Y_train, batch_size=512, nb_epoch=50, validation_split=0.05, verbose=1)
```

```
/home/varat/anaconda3/lib/python3.6/site-packages/keras/models.py:942: UserWarning: The `nb_epoch` argument in `fit` has been renamed `epochs`.
  warnings.warn('The `nb_epoch` argument in `fit` '
```

```
Train on 3475 samples, validate on 183 samples
Epoch 1/50
3475/3475 [==============================] - 3s 944us/step - loss: 7.7872 - val_loss: 6.8132
Epoch 2/50
3475/3475 [==============================] - 2s 498us/step - loss: 6.2169 - val_loss: 6.8919
Epoch 3/50
3475/3475 [==============================] - 2s 500us/step - loss: 5.5214 - val_loss: 5.7264
Epoch 4/50
3475/3475 [==============================] - 2s 482us/step - loss: 4.6849 - val_loss: 4.7646
Epoch 5/50
3475/3475 [==============================] - 2s 474us/step - loss: 3.8806 - val_loss: 3.6226
Epoch 6/50
3475/3475 [==============================] - 2s 497us/step - loss: 2.9888 - val_loss: 2.4607
Epoch 7/50
3475/3475 [==============================] - 2s 493us/step - loss: 2.4446 - val_loss: 2.2064
Epoch 8/50
3475/3475 [==============================] - 2s 491us/step - loss: 2.1474 - val_loss: 1.9311
Epoch 9/50
3475/3475 [==============================] - 2s 490us/step - loss: 1.8501 - val_loss: 1.5709
Epoch 10/50
3475/3475 [==============================] - 2s 475us/step - loss: 1.6355 - val_loss: 1.3805
Epoch 11/50
3475/3475 [==============================] - 2s 485us/step - loss: 1.3997 - val_loss: 1.1078
Epoch 12/50
3475/3475 [==============================] - 2s 503us/step - loss: 1.2262 - val_loss: 0.9279
Epoch 13/50
3475/3475 [==============================] - 2s 513us/step - loss: 1.0542 - val_loss: 0.7832
Epoch 14/50
3475/3475 [==============================] - 2s 498us/step - loss: 0.9369 - val_loss: 0.7451
Epoch 15/50
3475/3475 [==============================] - 2s 500us/step - loss: 0.8648 - val_loss: 0.7180
Epoch 16/50
3475/3475 [==============================] - 2s 488us/step - loss: 0.8081 - val_loss: 0.6389
Epoch 17/50
```

```
3475/3475 [==============================] - 2s 505us/step - loss: 0.7835 - val_loss: 0.7253
Epoch 18/50
3475/3475 [==============================] - 2s 491us/step - loss: 0.7858 - val_loss: 0.6530
Epoch 19/50
3475/3475 [==============================] - 2s 498us/step - loss: 0.7114 - val_loss: 0.6075
Epoch 20/50
3475/3475 [==============================] - 2s 484us/step - loss: 0.7013 - val_loss: 0.5255
Epoch 21/50
3475/3475 [==============================] - 2s 574us/step - loss: 0.6627 - val_loss: 0.5555
Epoch 22/50
3475/3475 [==============================] - 2s 564us/step - loss: 0.6375 - val_loss: 0.4784
Epoch 23/50
3475/3475 [==============================] - 2s 481us/step - loss: 0.6202 - val_loss: 0.4500
Epoch 24/50
3475/3475 [==============================] - 2s 470us/step - loss: 0.5633 - val_loss: 0.4295
Epoch 25/50
3475/3475 [==============================] - 2s 476us/step - loss: 0.5568 - val_loss: 0.3948
Epoch 26/50
3475/3475 [==============================] - 2s 484us/step - loss: 0.5459 - val_loss: 0.3649
Epoch 27/50
3475/3475 [==============================] - 2s 488us/step - loss: 0.5200 - val_loss: 0.3578
Epoch 28/50
3475/3475 [==============================] - 2s 498us/step - loss: 0.4902 - val_loss: 0.3428
Epoch 29/50
3475/3475 [==============================] - 2s 481us/step - loss: 0.4893 - val_loss: 0.3057
Epoch 30/50
3475/3475 [==============================] - 2s 675us/step - loss: 0.4663 - val_loss: 0.3167
Epoch 31/50
3475/3475 [==============================] - 2s 514us/step - loss: 0.4403 - val_loss: 0.2697
Epoch 32/50
3475/3475 [==============================] - 2s 525us/step - loss: 0.4343 - val_loss: 0.2614
Epoch 33/50
3475/3475 [==============================] - 2s 489us/step - loss: 0.4180 - val_loss: 0.2421
Epoch 34/50
3475/3475 [==============================] - 2s 491us/step - loss: 0.3963 - val_loss: 0.2342
Epoch 35/50
3475/3475 [==============================] - 2s 567us/step - loss: 0.3851 - val_loss: 0.2209
Epoch 36/50
3475/3475 [==============================] - 2s 646us/step - loss: 0.3794 - val_loss: 0.2063
Epoch 37/50
3475/3475 [==============================] - 2s 504us/step - loss: 0.3822 - val_loss: 0.2298
Epoch 38/50
3475/3475 [==============================] - 2s 482us/step - loss: 0.3548 - val_loss: 0.2001
Epoch 39/50
3475/3475 [==============================] - 2s 493us/step - loss: 0.3600 - val_loss: 0.1915
Epoch 40/50
3475/3475 [==============================] - 2s 490us/step - loss: 0.3483 - val_loss: 0.1856
Epoch 41/50
3475/3475 [==============================] - 2s 567us/step - loss: 0.3458 - val_loss: 0.1817
Epoch 42/50
3475/3475 [==============================] - 2s 531us/step - loss: 0.3527 - val_loss: 0.1829
Epoch 43/50
3475/3475 [==============================] - 2s 481us/step - loss: 0.3394 - val_loss: 0.2038
Epoch 44/50
3475/3475 [==============================] - 2s 538us/step - loss: 0.3248 - val_loss: 0.1812
Epoch 45/50
3475/3475 [==============================] - 2s 544us/step - loss: 0.3301 - val_loss: 0.1732
Epoch 46/50
3475/3475 [==============================] - 2s 542us/step - loss: 0.3209 - val_loss: 0.1974
Epoch 47/50
3475/3475 [==============================] - 2s 494us/step - loss: 0.3207 - val_loss: 0.1747
Epoch 48/50
3475/3475 [==============================] - 2s 498us/step - loss: 0.2996 - val_loss: 0.1649
Epoch 49/50
3475/3475 [==============================] - 2s 517us/step - loss: 0.2992 - val_loss: 0.1623
Epoch 50/50
3475/3475 [==============================] - 2s 484us/step - loss: 0.2906 - val_loss: 0.1615
```

Out[14]:

```
<keras.callbacks.History at 0x7f897c241da0>
```

Evaluate the result

In [15]:

```
mse_test = model.evaluate(X_test, Y_test, verbose=1)
print('\n mean squared error(MSE) on the test data set is %.3f over %d test samples.' %(mse_test, len(Y_test)))
```

```
1219/1219 [==============================] - 0s 294us/step

 mean squared error(MSE) on the test data set is 0.519 over 1219 test samples.
```

Get the predicted values

In [16]:

```
predicted_values = model.predict(X_test)
num_test_samples = len(predicted_values)
predicted_values = np.reshape(predicted_values, (num_test_samples, 1))
```

Plot the results

In [17]:

```
fig= plt.figure()
plt.plot(Y_test + load_mean)
plt.plot(predicted_values + load_mean)
plt.xlabel('hours (hrs)')
plt.ylabel('Electricity Load (*1e5)')
plt.show()

# Save the plot into a jpg file
fig.savefig('Forecasted_Load_Results.jpg', bbox_inches='tight')

# Save the results into txt file
test_result = np.vstack((predicted_values + load_mean))
np.savetxt('Forecasted_Load__Results.txt', test_result)
```