Time Series Analysis:

Load Forecasting Track of Global Energy

Yijing Yang
Department of Computer Science
University College London
London, WC1E 6BT
yijing.yang.15@ucl.ac.uk

Xinyi He
Department of Computer Science
University College London
London, WC1E 6BT
xinyi.he.15@ucl.ac.uk

Ying Wen
Department of Computer Science
University College London
London, WC1E 6BT
ying.wen@cs.ucl.ac.uk

Github link of our project: https://github.com/wenying45/time_series_prediction

1 Introduction

This project is related to the load forecasting track of global energy, a hierarchical forecasting problem: backcasting and forecasting hourly loads (in kW) for a US utility with 20 zones. We are going to backcast and forecast at both zonal level (20 series) and system (sum of the 20 zonal level series) level, totally 21 series.

For this project, the load of energy is mainly subjected to temperatures and calendar information. One of our data sets is hourly load history of 20 zones, the other one is hourly temperature of 11 stations. However, there is no indication to tell us which station is related to each zone. Therefore, our main task of this project is to (1) find a suitable model to define the relation between zones and stations. (2) use temperatures, calendar information and other related features as inputs to predict the load of energy.

In this paper, we perform two models. One is XGBoost and the other one is a neural network model, LSTM model. For XGBoost, we do feature engineering, including transforming categorical attributes and creating features for time lag effects. After training the XGBoost, the performance is not as ideal as imaged. Therefore, we turn to LSTM model. Different from XGBoost, LSTM model does not require us to match stations with zones to choose the right temperatures as the feature. So LSTM costs less time and gains higher RMSE performance. For LSTM model, we try 2-3 layers with different inputs. As a result, we reach the best performance with 3 layers and history load as input.

2 Background

2.1 eXtreme Gradient Boosted Trees

XGBoost is short for “Extreme Gradient Boosting”. It is used for supervised learning problems, where we can use the training data to do the prediction. The model of XGBoost is tree ensembles and the tree ensemble model is a set of classification and regression trees. We need to sum the prediction of multiple trees rather than focusing on a single tree.

We can assume that we have the following objective function which is made up with the training loss and the regularization.

$$\mathrm{Obj} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i)$$

The training loss measures how predictive our model is on training data and the regularization term controls the complexity of the model and helps us to prevent over fitting.

In XGBoost, we define the complexity as

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Here T is the number of leaves in the tree, w is the vector of scores on leaves, λ is the leaf weight penalty parameter and γ is the tree size penalty parameter.

To measure how good a tree structure $q(x)$ is, we get the following equation after few steps:

$$\mathrm{Obj}^* = -\frac{1}{2}\sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

where

$$G_j = \sum\nolimits_{i \in I_j} g_i = \partial_{\hat{y}^{(t)}} l(y_i, \hat{y}_i^{(t-1)})$$

$$H_j = \sum\nolimits_{i \in I_j} h_i = \partial^2_{\hat{y}^{(t)}} l(y_i, \hat{y}_i^{(t-1)})$$

Because it's hard to list all possible trees and get the best one, it's wiser to optimize one level of the trees at a time. We can split a leaf into two leaves, and it gains the score as follows:

$$\mathrm{Gain} = \frac{1}{2}\left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}\right] - \gamma$$

where L and R stand for the left and right leaves. Sometimes if the gain is smaller than γ, we'd better not add this part.

Tree boosting is a significant kind of machine learning algorithms that are widely used now. Tianqi Chen and Carlos Guestrin (2015) [5] say in the paper that tree boosting algorithms are quite limited to large scale datasets and then introduced a reliable and distributed machine learning system that can scale up tree boosting algorithms. This system can not only tackle tens of millions of datasets on a single machine, but also scale beyond billions of datasets in distributed setting.

Brian Knott et al. (2015) [11] used extreme boosted trees to train the gradient boosting models and tried to use parameter optimization to get the best solution. They used several models to do the time series forecasting and gradient boosting method with XGBoost package. Finally, gradient boosting performs the best. Maksim Korolev and Kurt Ruegg (2015) [10] also used gradient boosted trees to do predictions based on a 6-week time series dataset and got a better result after doing optimization to modify the hyperparameters of the gradient boosted trees.

2.2 Recurrent neural network

Recurrent neural network model is a class of artificial neural network (ANN). It makes an internal state of the network which allows it to exhibit dynamic temporal behavior. RNN, unlike feed forward neural networks, can use the internal memory to process arbitrary sequences of inputs, making it possible to be applied to many different fields.

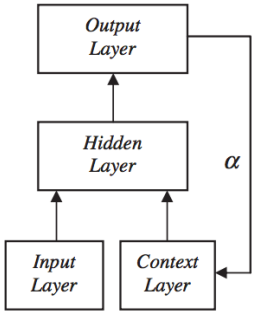


Figure 1 Recurrent neural network

J. Vermaak and E.C. Botha (1998) [7] used RNN to do short-term load forecasting. The recurrent network successfully captured the dynamic behavior of the load, and led to a compact and natural internal representation of the temporal information contained in the load profile. Moreover, the RNN architecture facilitates a training method of no more than feed forward complexity, making a faster optimization.

Qing Cao et al. (2012) [6] presented a comparative analysis of the wind speed prediction of univariate and multivariate ARIMA models with their RNN counterparts. The analysis used wind speed time series datasets from the same tower location for different heights above ground level. The results showed that the RNN models outperform the ARIMA models and the multivariate models perform better.

Recurrent neural network (RNN) has many architectures, one of which is called Long Short-Term Memory (LSTM) network. LSTM is now commonly used by many researchers. LSTM RNNs prevent backpropagated errors from vanishing or exploding (Sepp Hochreiter, 1991) [8]. Instead, errors can flow backwards through unlimited numbers of virtual layers in LSTM RNNs unfolded in space. That is, LSTM can complete deep learning tasks. (Jürgen Schmidhuber, 2015) [9] LSTM's learning algorithm is local in space and time, and its computational complexity per time step and weight is O(1). (Hochreiter & Schmidhuber, 1991) [9] Unlike the traditional RNNs, an LSTM network is good at learning from history and experience to classify, process and predict time series when long time lags exist. That's why LSTM performs better than alternative RNNs.

3 Methodology

3.1 Data preprocessing

Before we train the model and use the model to make prediction, we apply some methods to do data preprocessing for the raw history data, which includes time, history temperature and history load.

Mean subtraction, also called zero-centered, which is one of the most common data preprocessing method. It interprets the origin data around the origin point along every dimension by subtracting the mean of each individual dimension. For example, we calculated the mean load of each zone, then make each zones' load subtract corresponding mean.

Normalization, which means normalizing the data in every dimension to an approximated

scale. We can implement it by dividing each dimension's standard deviation after the data is processed by the mean subtraction [1]. For example, we calculate the standard deviation of load of each zone, then make each zones' zero-centered load divided by corresponding standard deviation.

Then following figure illustrates the processes of mean subtraction and normalization:

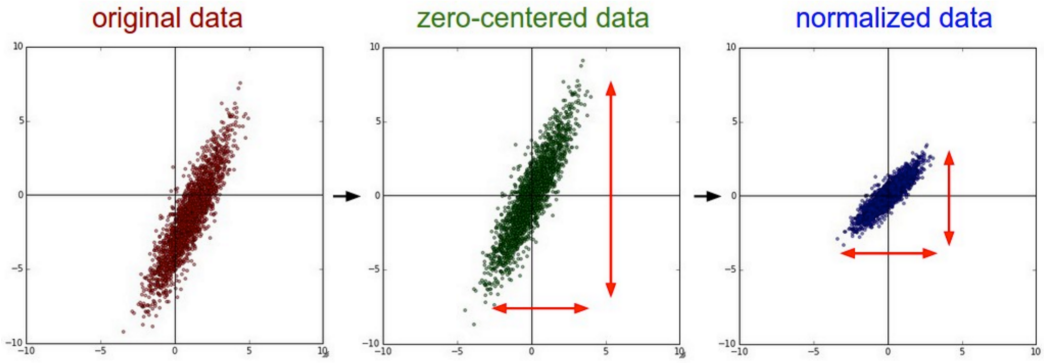


Figure 2 The data normalization processes [1]. Left: raw data. Middle: zero-centered data by subtracting the mean in each dimension. Right: each dimension is additionally scaled by its standard deviation.

Besides, a log transformation is applied for load history data whose data range varies from hundreds to hundreds thousand, and this will make models hard to be trained. After the log transformation, the load values are scaled to a reasonable domain, then use normalization method we mentioned above to normalize the data. After the prediction is made, we use the reverse processes to recover the predicted data to original value space.

3.1 XGBoost

Motivation

XGBoost model is a kind of tree models, with more efficiency and higher accuracy. The model can process large data sets more quickly than others. This is the main reason we choose XGBoost. Our resource data includes time series so that we need to consider time lag effects. Moreover, most of the features are categorical variables such as zone_id, year, month, day and hour. These categorical variables should be converted to dummies for the prediction. Thus, after processing the resource data, we would have much more features. XGBoost model can help us handle such a big data set more efficiently and quickly.

Feature Engineering

There are two parts for feature engineering: 1) transforming categories to dummies; 2) building features for time lag effects.

In our resource data, there are 5 categorical variables: year, month, day, hour and zone_id. We use 'get_dummy' in Python to convert the variables to binary ones. Take 'year' as an example, there are 5 levels for 'year' in all: '2004', '2005', '2006', '2007' and '2008'. After processed, there would be five new columns: 'Year_2004', 'Year_2005', 'Year_2006', 'Year_2007' and 'Year_2008'. If it is true, then the value for the column would be 1. Otherwise, the value would be 0.

For time lag effects, we firstly build 24 features for lags from 1 to 24 since records of these 24 hours have more effect than others. With the lagged loads as features, we use XGBoost to check the feature importance.

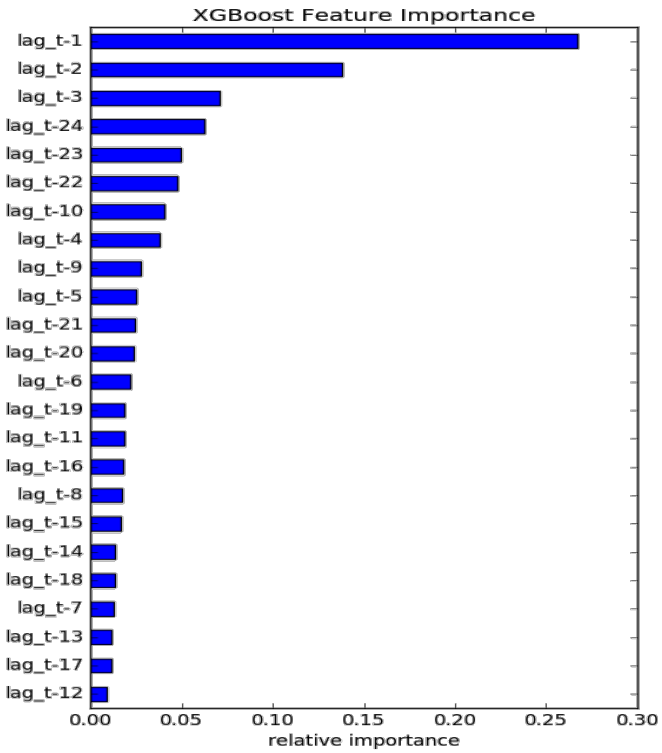


Figure 3 Feature importance for lagged loads with XGBoost model

As Figure 2 shows, the most important features are lag_t-1, lag_t-2, lag_t-3 and lag_t-24. Therefore, for the prediction, we finally choose these 4 features as time lag effects.

Temperature Matching

After feature engineering, we get all the features for the prediction, except temperature. In our resource data, we have temperatures of 11 stations. However, it is not clear that which temperatures are related to each zone. To solve the problem, we train the model for each zone with one of the stations every time and compare the results of MSE. We consider temperatures with the smallest MSE as correct temperatures for the zone. Table 1 gives the results of the match:

Table 1 Matches for zones and stations based on MSEs

Zone	Station	Zone	Station
1	2	11	1
2	11	12	3
3	11	13	2
4	9	14	4
5	11	15	8
6	11	16	3
7	11	17	6
8	2	18	3
9	6	19	6
10	1	20	11

However, matching stations and zones with this kind of methods may cause errors. And errors will lead to large RMSE when we compare the predictions with actual data. This is what we should improve in the future.

3.1 Stateful recurrent neural network

Motivation

Different from the feed forward neural networks, recurrent neural networks (RNNs) have self-loop connection. This allows the network to store an internal state and consequently process sequences of self loop connection, which could be expended as a sequence. This feature makes the RNNs naturally accept the sequence inputs [2]. Besides, since RNNs can store the memory of previous activities, history memory can be stored and used to make future prediction.

Unfortunately, the simple RNN suffers from gradient vanish issues, in an other word, the simple RNN owns weak history memory ability and it will make model training harder. Therefore, we choose Long Short Term Memory (LSTM) network, which has same structure but more complex unit to capture long term memory [3].

In addition, one challenge for this task is that we need to figure the relationship between 11 stations and 20 zones, which will bring much more features engineering and data processing work. But one of characteristics of neural network is strong feature learning and fitting ability. It can reduce lots of manual work, maybe with better performance as well.

Stateful LSTM Model

As our task is time series prediction, the common LSTM/RNN will reset the unit memory after each prediction, but the previous memory is useful for the current time's prediction. To solve this issue, we use stateful LSTM which will keep the status of units after each time step's prediction. The state of units only will be reset after each epoch and that means all time steps in training set is trained and predicted for a once.

Generally, the deeper neural networks will have better fitting ability but not easy to be well trained. We train the models with 2-3 layers LSTM which will achieve the balance between model performance and training time. In common case, mini-batch gradient decent method is used to train the neural network, so the batch size selection is important. Based on our experiments, the small batch size will make lower loss but increase training time. At some times, lower batch size seems to will bring over fitting problem, so we choose batch size as 2 which achieved better prediction performance on testing data with reasonable training time. As for the adaptive training methods, in practice Adam is currently recommended as the default algorithm to use in many neural networks, interestingly, in recurrent neural work, although the Adam can decrease convergence time, RMSProp can lead to better minimal result in the most of time.

The input of our stateful LSTM model is preprocessed time, history temperature data and previous week's load data at some time in a week. The motivation that we choose previous week's load data as input is that we want to enforce the history memory of the most recent time with same week day and hour to make better short term prediction. We also test the model without history load as input, which has achieved slightly bad performance. The output of the model is the loads of 20 zones.

To more intuitively and vividly illustrate our multiple layers stateful LSTM model, the following figure shows the structure and processes of the stateful LSTM model:

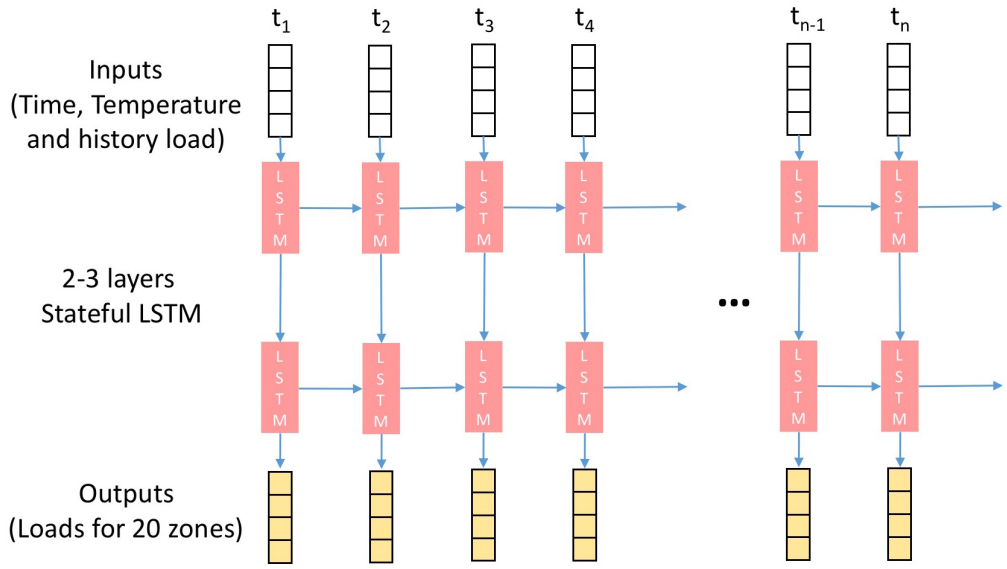


Figure 4 The structure and processes of multiple layers stateful LSTM neural netwok model

4 Experiment

Data description

Data (loads of 20 zones and temperature of 11 stations) history ranges from the 1st hour of 2004/1/1 to the 6th hour of 2008/6/30. Given actual temperature history, and 8 weeks in the load history are set to be missing.

Because the original dataset does not provide some data, then we fill the missing data with history average and take the last week data which is provided with real data as testing set [4]. Then we split the total data into 2 parts: training set from 2004/1/1 to 2008/06/23 and one week testing data from 2008/06/24 to 2008/06/30.

As for baseline, we choose two history averages as baseline to evaluate our models' performance: history average at same time (means same date and same hour) of past four years and previous week's history data at same time (means the most recent same week day with same hour).

Experiment results

The details of results are listed in the following Table 2. One thing needs to be mentioned is that the zone 21 which is load sum of the other 20 zones is included when calculate the root mean square error (RMSE) on testing data from 2008/06/24 to 2008/06/30:

Besides, the following figure 4 shows the fitting and forecasting result for 21 zones of our best model, which more vividly displays our model performance. More figures are appended at appendix section includes 4 months fitting figure of our best model and the figures of other models.

From the figure, we can find that in most of times, our model has great fitting and prediction performance, especially, in zone 21, the sum load of 20 zones. For this zone, our model almost to have perfect fitting and prediction. It means our model has good ability to capture the overall trend of 20 zones.

Table 2 The RMSE of different models, which is calculated on testing data from 2008/06/24 to 2008/06/30

Model		RMSE
History Average Baselines	Average by previous years	167,462
	Average by previous week	227,129
XGBoost [5] [Github Link]	N/A	50,456
Stateful LSTMs [Github Link]	2 layers without history load as input	28,919
	2 layers with history load as input	32,957
	3 layers with history load as input	**19,231**

However, there are still some issues for our model. In the figures, the blue line represents the real load of each zone. We can see that in many zones (zone 2,3,4,5,6,7,8,10), the loads around 2016/06/28 have a significant increase, compared with same time of previous week. Although our model forecast higher load compared with the real loads and predictions of previous week at same time, it still does not totally handle such rapid changes. Besides, the zone 9 needs to be discussed individually. From the figure of zone 10, it is obviously that real load data (blue line) has sharp changes at many times and periodic feature is less significant, compared with the other zones. Both of the reasons cause our model has really bad performance on this zone.

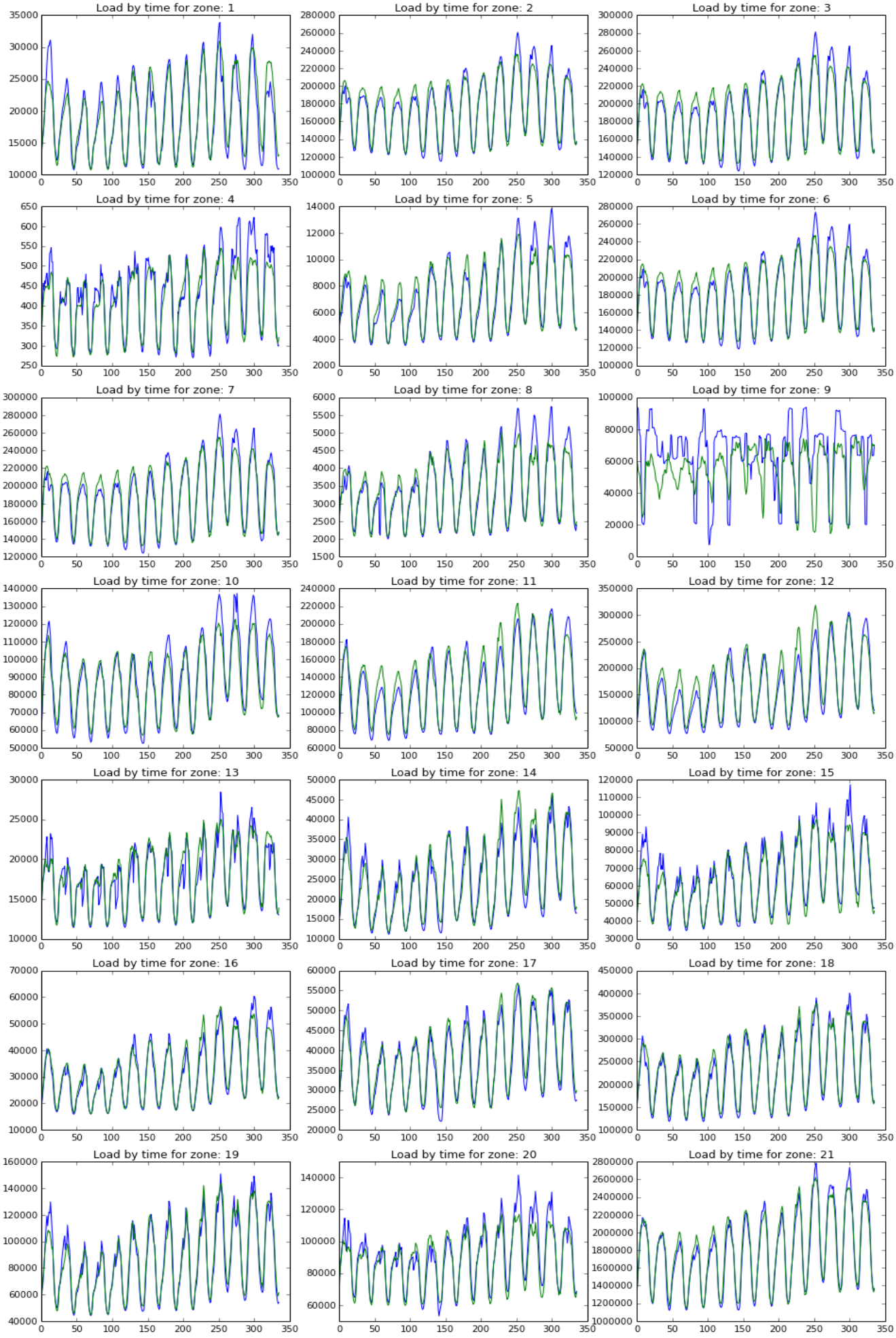

Load by time for zone: 1
Load by time for zone: 2
Load by time for zone: 3
Load by time for zone: 4
Load by time for zone: 5
Load by time for zone: 6
Load by time for zone: 7
Load by time for zone: 8
Load by time for zone: 9
Load by time for zone: 10
Load by time for zone: 11
Load by time for zone: 12
Load by time for zone: 13
Load by time for zone: 14
Load by time for zone: 15
Load by time for zone: 16
Load by time for zone: 17
Load by time for zone: 18
Load by time for zone: 19
Load by time for zone: 20
Load by time for zone: 21

Figure 5 The three layers stateful LSTM with history load as input model's fitting and forecasting results from 17/06/2008 to 30/06/2008. In the first half, 17/06/2008 to 23/06/2008 is training fitting data; The second half from 24/06/2008 to 30/06/2008 is testing forecasting data. Besides, the blue line represents expected data and the green line represents forecasting data.

5 Conclusion & Future work

In this report we use two models for Kaggle energy forecasting task, where the stateful LSTM model achieves the best performance. But our models still have some issues that can not handle: 1）The most important thing is to find a reasonable method to match stations and zones. In the future work, it is suggested to seek for more related literature. Moreover, to increase the performance, people can try different models to check the average MSE of each station for each zone. 2) The current codes we offer for XGBoost cost long time to calculate MSEs, which are used to match zones and stations. It is better to improve the efficiency of the codes. Besides, XGBoost has often been shown to be an effective method in Kaggle competition, both in regression and classification. Although its performance is not as ideal as LSTM model, but maybe we can apply ensemble method on this task, and use the different models to form a better model.

Reference

[1] Cs231n.github.io. (2016). CS231n Convolutional Neural Networks for Visual Recognition. [online] Available at: http://cs231n.github.io/neural-networks-2/ [Accessed 15 Apr. 2016].

[2] Kaastra, Iebeling, and Milton Boyd. "Designing a neural network for forecasting financial and economic time series." Neurocomputing 10.3 (1996): 215-236.

[3] Gers, Felix A., Jürgen Schmidhuber, and Fred Cummins. "Learning to forget: Continual prediction with LSTM." Neural computation 12.10 (2000): 2451-2471.

[4] Taieb, Souhaib Ben, and Rob J. Hyndman. "A gradient boosting approach to the Kaggle load forecasting competition." International Journal of Forecasting 30.2 (2014): 382-394.

[5] Chen, Tianqi, and Tong He. "xgboost: eXtreme Gradient Boosting." R package version 0.4-2 (2015).

[6] Qing Cao, Bradley T. Ewinga, Mark A. Thompson. "Forecasting wind speed with recurrent neural networks." European Journal of Operational Research. Volume 221, Issue 1, 16 August 2012, Pages 148–154.

[7] J. Vermaak, E. C. Botha. "Recurrent neural networks for short-term load forecasting." IEEE Transactions on Power Systems. Volume 13, Issue 1, Feb 1998, Pages 126-132.

[8] Sepp Hochreiter (1991), Untersuchungen zu dynamischen neuronalen Netzen, Diploma thesis. Institut f. Informatik, Technische Univ. Munich. Advisor: J. Schmidhuber.

[9] Jürgen Schmidhuber (2015). Deep learning in neural networks: An overview. Neural Networks 61 (2015): 85-117.

[10] Maksim Korolev, Kurt Ruegg (2015). "Gradient Boosted Trees to Predict Store Sales."

[11] Brian Knott, Hanbin Liu, Andrew Simpson. (2015) "Predicting Sales for Rossmann Drug Store." CS229 Final Paper.

Appendix

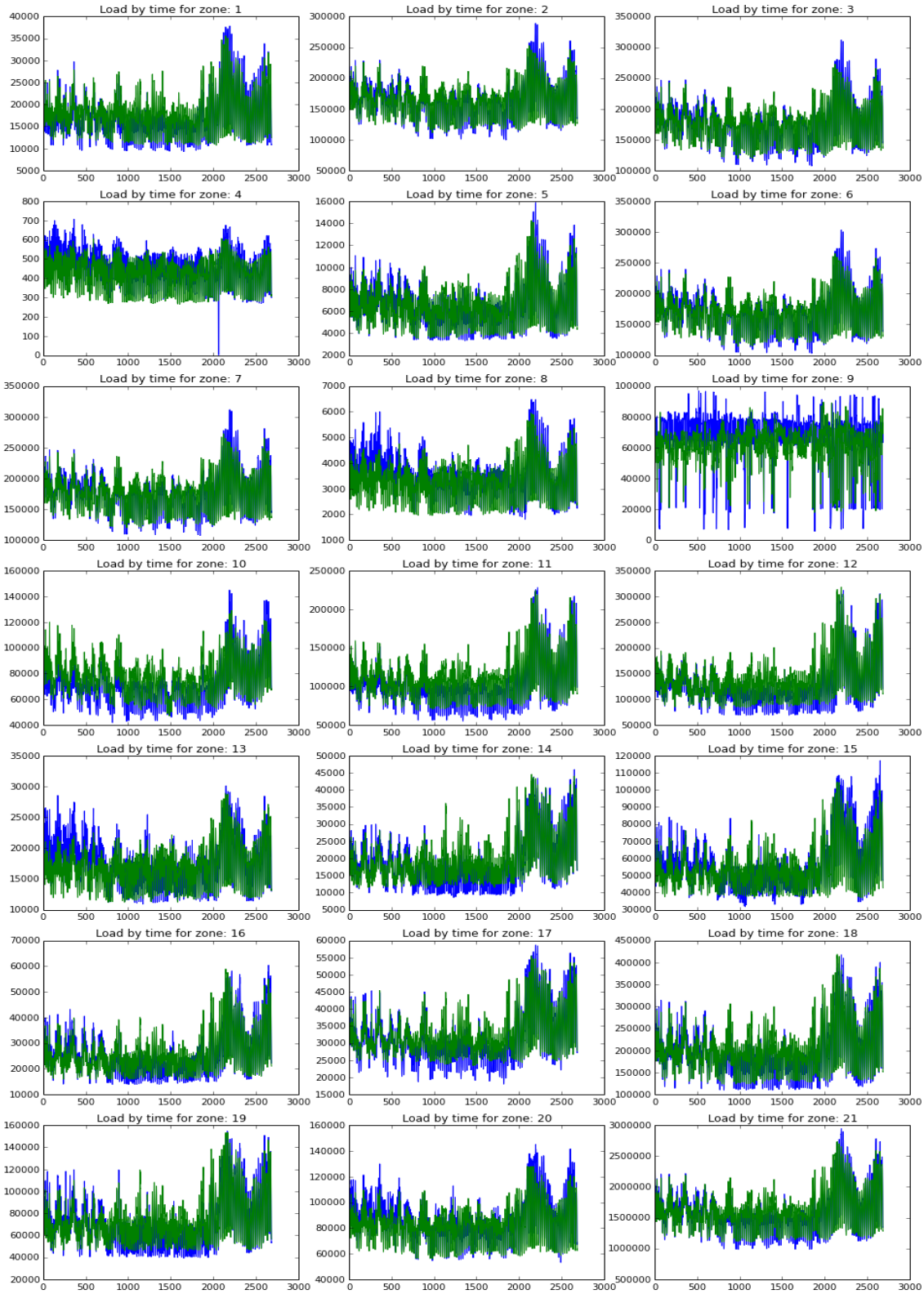
Appendix 1

The results of MSE for each zone

	zone1	zone2	zone3	zone4	zone5	zone6	zone7	zone8	zone9	zone10
station1	21270952.81	392999315.47	433609966.24	155327.34	3107361.72	473633123.99	433609966.24	326985.00	318402756.37	2138115430.32
station2	3927653.90	249274255.85	274159245.70	168043.33	1711032.84	263592376.42	274159245.70	157647.88	310295108.66	2647096352.07
station3	18121670.57	280670680.94	318407590.91	156496.74	2186285.17	269190145.73	318407590.91	268429.33	377846523.96	2617999483.97
station4	22002925.58	420464942.51	485972255.22	163957.23	3543525.82	501647250.14	485972255.22	328184.82	291659701.50	2555747620.37
station5	19475431.26	362733863.32	497630909.72	151713.83	2413399.33	455931423.43	497630909.72	294851.35	311736911.92	2658368073.82
station6	8419631.40	221252753.96	255603428.42	168533.45	2562803.00	260082638.10	255603428.42	253689.18	289928971.03	2665798574.75
station7	12542541.83	284793706.57	351064111.74	168609.25	1590116.06	390149162.76	351064111.74	205091.80	326421974.88	2579333664.60
station8	18832205.34	305123177.81	383132785.54	165972.19	3155075.56	331033477.93	383132785.54	252298.06	329569987.64	2601950167.21
station9	15809877.50	287098076.38	357854719.90	147877.59	2162558.32	350951883.86	357854719.90	214043.94	468226531.82	2662031569.46
station10	9878435.28	237228090.67	273755473.74	169944.55	2516188.51	248267388.67	273755473.74	237374.21	292716362.31	2638980411.28
station11	11106709.43	206671079.10	246250869.90	165121.26	1342579.76	159214491.43	246250869.90	175156.48	382525882.94	2671652631.33
	zone11	**zone12**	**zone13**	**zone14**	**zone15**	**zone16**	**zone17**	**zone18**	**zone19**	**zone20**
station1	182652317.50	816121302.62	21025961.78	7254727.99	239581180.54	9030183.88	11079495.08	1438385165.85	583721115.68	3280938785.66
station2	1559040211.85	4602955426.22	6465778.66	21780740.73	281307666.61	57717641.67	15828984.77	3077555521.24	647232114.56	3023055272.30
station3	190481363.29	373901565.00	17461109.57	8050108.45	242281771.88	7087420.66	9785031.75	1302939299.83	568582995.74	3259010216.79
station4	339817228.66	1719146934.31	22646926.40	6293393.20	163857704.09	11395228.72	8758832.70	2140505312.56	525984584.74	3070343253.62
station5	291802216.56	636070932.52	17628211.84	8420109.94	281456200.65	7772363.26	11899343.97	2038107924.52	640658283.58	3300624823.01
station6	605460607.57	3405121294.16	11653729.48	7776772.41	164195332.03	21783161.11	5424060.03	1872511059.30	366102864.18	3182229323.25
station7	642436518.17	2251249556.03	10061744.71	8339379.88	224540483.30	15589746.28	8119351.59	1999377143.34	455674934.64	3143042210.83
station8	374915100.22	1551764031.91	19228820.49	6794406.07	142981255.08	9428809.06	6859097.07	1691966895.37	413035240.38	3216596835.24
station9	454979694.25	1923317299.36	13342301.72	7309943.17	240972218.88	11032435.79	8065537.54	1955689498.21	497307390.45	3259491106.34
station10	586971237.99	2753304487.23	10441142.01	8491224.30	149923900.44	18358360.34	5703442.74	1993838400.53	388109017.38	3172313365.60
station11	1352435109.27	3803719432.36	7005695.56	15355356.23	344537369.02	33793895.04	14214847.0688003	2913478856.52	726446323.20	2757424096.70

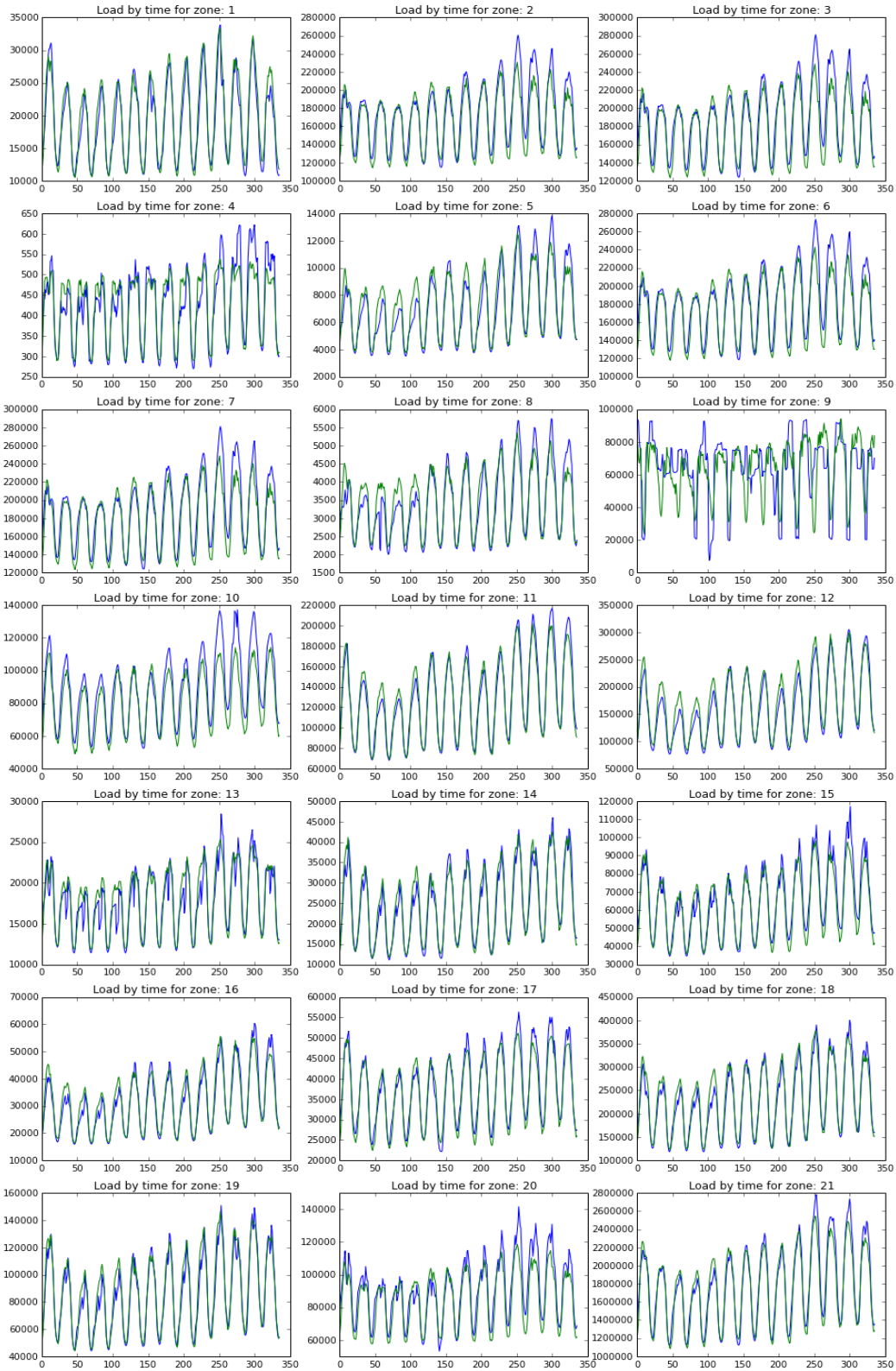
Appendix 2

The 3 layers stateful LSTM with history load as input model, 4 months' figure from 2008/03/01-2008/06/30. Note, this is our best model.



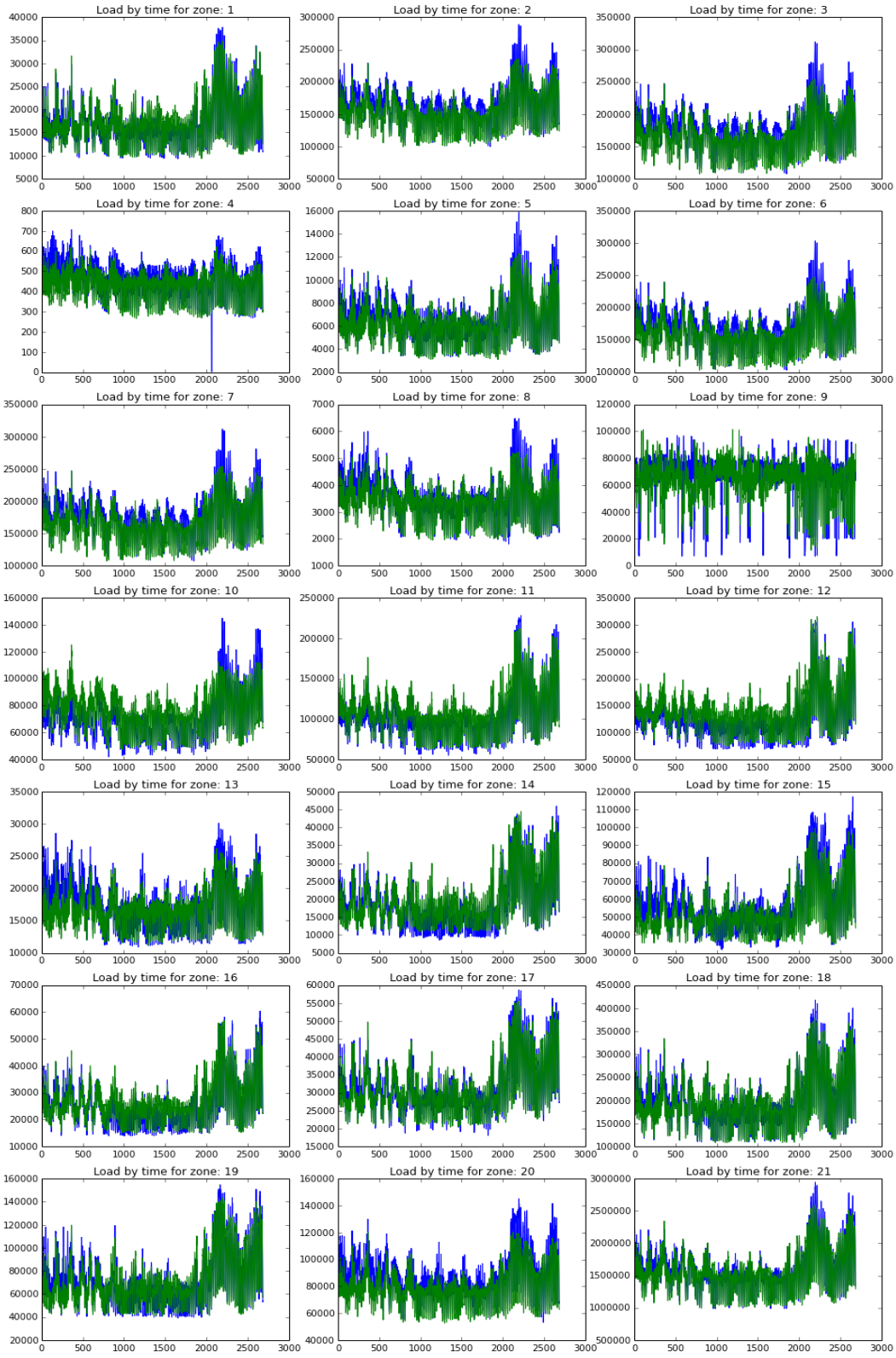
Appendix 3

The 2 layers stateful LSTM without history load as input model, 2 weeks' figure from 2008/06/17-2008/06/30.



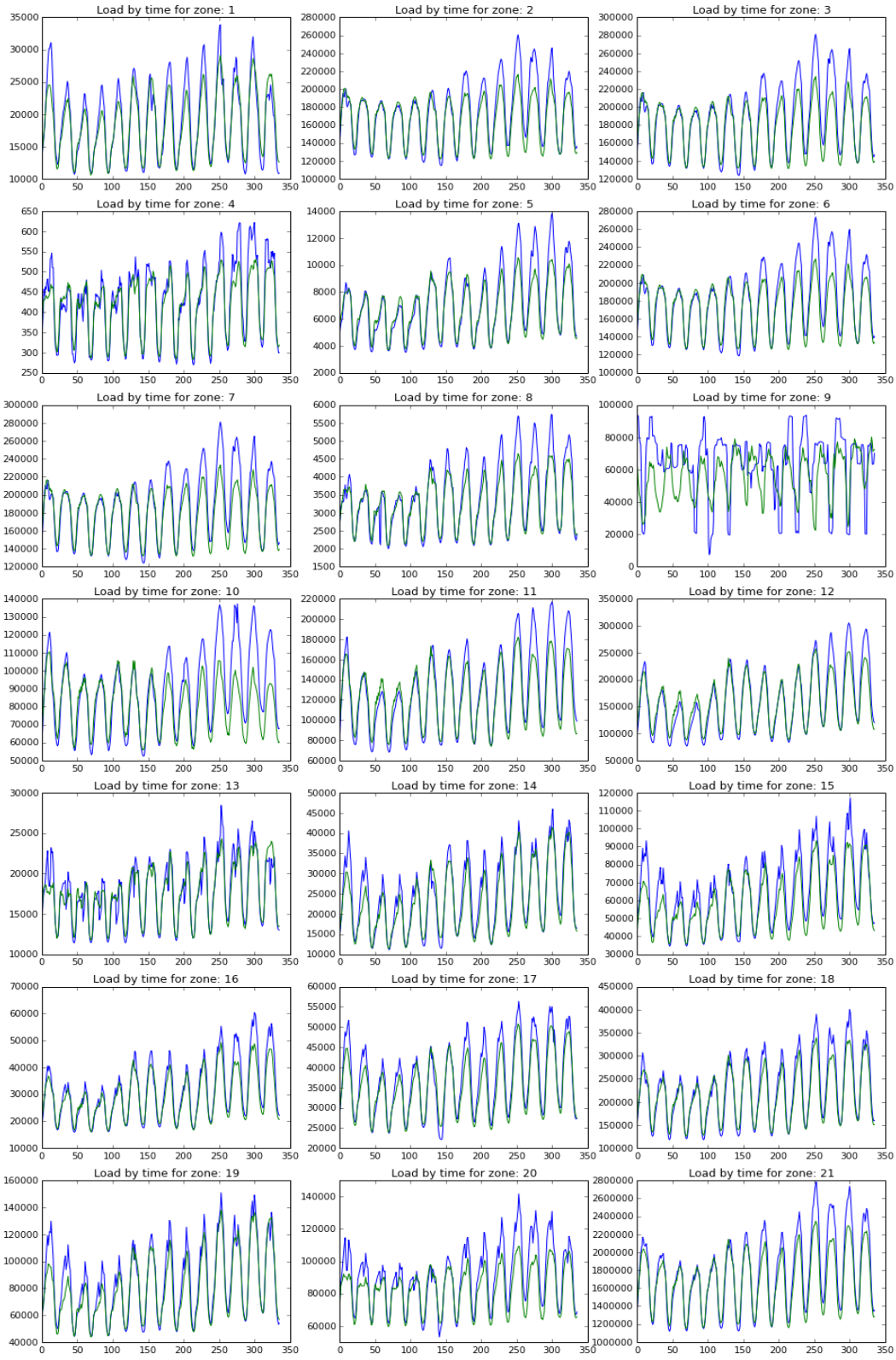
Appendix 4

The 2 layers stateful LSTM without history load as input model, 4 months' figure from 2008/03/01-2008/06/30.



Appendix 5

The 2 layers stateful LSTM with history load as input model, 2 weeks' figure from 2008/06/17-2008/06/30.



Appendix 6

The 2 layers stateful LSTM with history load as input model, 4 months' figure from 2008/03/01-2008/06/30.