Traffic Flow Prediction With Big Data Using A Deep Learning

Final Report

Lin Lyu

1. Problem Definition and Background

Traffic flow prediction largely relies on historical and real-time traffic data collected from various sensor sources, including sensor coils, radars, cameras, mobile global positioning system, crowdsourcing, social media, etc. With the extensive application of traditional traffic sensors and emerging traffic sensor technologies, and the explosive growth of traffic data, transportation management and control is becoming more and more data-driven[1]. Although there are many traffic flow prediction systems and models, most of them adopt shallow traffic models, like one hidden layer neural network, which still have some shortcomings. This inspires us to rethink the traffic flow prediction based on deep structure models, which have such abundant traffic data. And it is also worth to try different models to predict traffic flow and compare these models performance, considering the number of parameters, the training speed, the evaluation metrics and etc., to choose a best model.

2. Solution Method

As a traffic flow process is complicated in nature, deep learning algorithms can represent traffic features without prior knowledge, which has good performance for traffic flow prediction. Some papers proposed deep-learning-based traffic flow prediction methods, a stacked autoencoder (SAE) model is used to learn generic traffic flow features, and it is trained in a layerwise greedy fashion. The traffic flow prediction problem can be stated as follows. Let Xit denote the observed traffic flow quantity during the time interval at the ith observation location in a transportation network. Given a sequence {Xit} of observed traffic flow data, i = 1, 2, . . . , m, t = 1, 2, . . . , T , the problem is to predict the traffic flow at time interval (t + Δ) for some prediction horizon Δ[1].

Here I used a timesteps of 12 and then use the 12 timesteps to predict the next timestep. There are three NN model I plan to implement through python pytorch package:

Model 1: GRU(Gated Recurrent Unit), RNN

Model 2: LSTM(Long Short-Term Memory), RNN[2]

Model 3: Self Predict Model

3. Data Used

Data from the Caltrans Performance Measurement System (PeMS) database are collected every 30s from over 15000 individual detectors, which are deployed statewide in freeway systems across California. The collected data are aggregated 5-min interval each for each detector station. Website: http://pems.dot.ca.gov/

4. Research Process

4.1 Understand the Data

The data are collected from multiple detectors on freeways. Each row represents one timestep, having data from different detectors on this freeways. I used interstate-5 North

with miles of 13.8. I used February, March from PeMS as training data, and April as test data. And I also plot some daily trend and weekly trend to get some insight. The results show that, the trend is regular and we could use this character to predict future traffic flow.

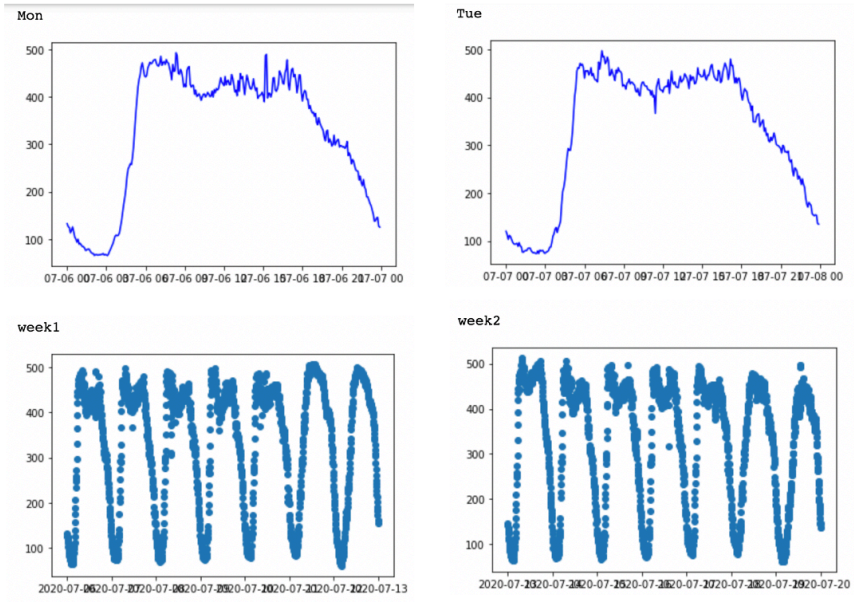


Figure 1. Daily, weekly trend

We could see from the above figure, the daily trend and weekly trend both have a specific trend. So it is reasonable to predict traffic trend.

4.2 Data Preprocessing

The first step is to read in the train data and test data csv files using Pandas. Then I calculated the average flow for I-5 North across all stations. Considering data normalization problem, the calculated maximum average flow is 543 per 5 minutes and minimum average flow is 57 per 5 minutes so it is necessary to standardize data before modeling. Because when attempting to use DL algorithms to find trends in the data by comparing features of data points, there is an issue when the features are on drastically different scales[3]. Variables that are measured at different scales do not contribute equally to the model fitting & model learned function and might end up creating a bias[4]. The goal of normalization is to make every datapoint have the same scale so each feature is equally important.

So we need to choose normalization method to standardize data. There are two often used methods one is Min Max and another is Z-score. Here I chose to use Min Max, because according to the scatter plot, we can see all the data distribution is reasonable, we do not need to worry the outliers too much, while Z-score is handling outliers well but would normalize the data with different scales.

Also, keeping feature scale with 0 to 1 could help the model learn more quickly, because it could decrease the internal covariate shift and it can prevent the gradient explosion. Covariate shift means the distribution of the features is different in different parts of the training/test data, in this question, it means the average flow varies very different across the whole day.

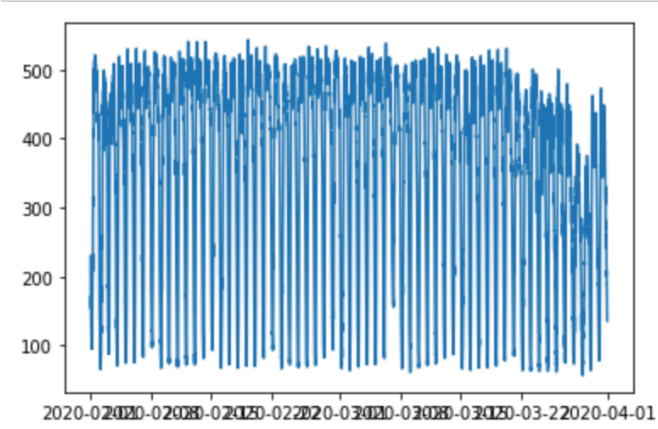


Figure 2. Data trend before standardize

The image below shows the same house data normalized using min-max normalization.

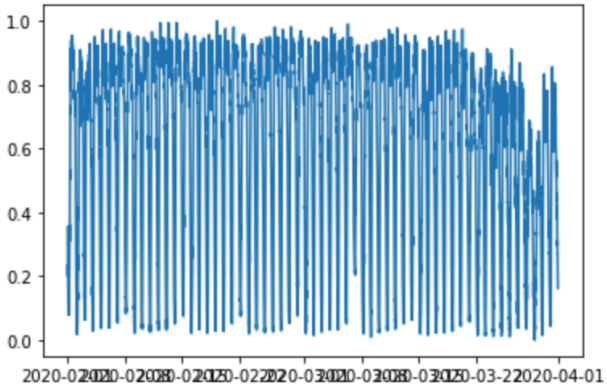


Figure 2. Data trend after standardization to 0 to 1

Also, when preprocessing data for using in the model, I used window length of 13, and then begin to split the train and test dataset. For a more detailed process, we can see in the following:

If we assume every 13 timesteps is identical independent distribution, we can then split the data and create mini-batch data for model. For train data: the original dimension is (17268,); after processing to make mini batch data, use a window of 13 timesteps to scan the whole timesteps with one step further for next scanning, we can get train data of shape (17256,12) and train label of shape(17256,1). (17256,13) means that we have 17256 samples and each sample with a dimension of 12 timesteps and each sample is i.i.d, and for each sample, we split each sample and use the first 12 as train to predict the 13 timestep. And then I did the same thing for test data, the shape changes from (8639,) to (8627,12), the test data shape, and got test label of shape (8627,1).

Table 1. Training/Test Data shape after preprocess

Train sample shape	Train data	Train label
(17256,13)	(17256,12)	(17256,1)
Test sample	Test data	Train label
(8627,13)	(8627,12)	(8627,1)

4.3 RNN, LSTM vs GRU

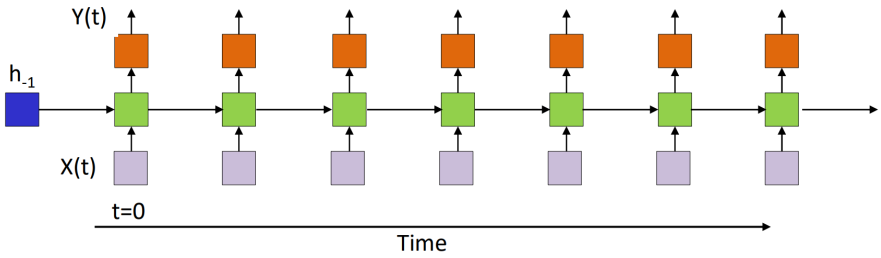


Figure 3. Overall RNN structure

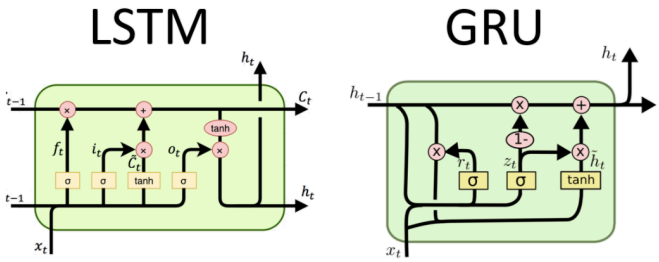


Figure 4. LSTM and GRU units structure

LSTM (Long Short Term Memory): LSTM has three gates (input, output and forget gate). GRU (Gated Recurring Units): GRU has two gates (reset and update gate).

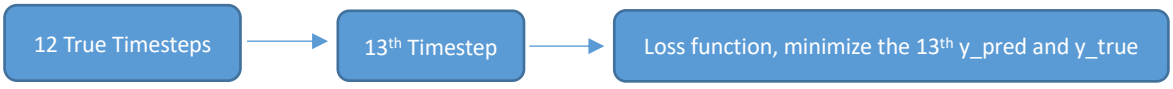
GRU use less training parameters and therefore use less memory, execute faster and train faster than LSTM's whereas LSTM is more accurate on dataset using longer sequence.

In short, if sequence is large or accuracy is very critical, we should use LSTM whereas for less memory consumption and faster operation go for GRU. If you do not have much floating point operations per second (FLOP's) to spare switch to GRU.

4.4 GRU for Predicting Time Series Data

Part 1:

We could see from the following flowchart, here each sample is consisted of 12 timesteps. In the training process, after each sample, the loss function is used to minimize the difference between the 13the predict and 13th true label. The cycle repeats using this rule, every time we put 12 true timesteps and then predict the 13th. Each pair sample (12timesteps, 13thtimestep) is preprocessed , and there is no scanning process. In part 2, I will introduce the scanning process.



We could see the model structure and number of parameters of the GRU model in the following two figures.

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru_16 (GRU)                 (None, 12, 64)            12864
_________________________________________________________________
gru_17 (GRU)                 (None, 64)                24960
_________________________________________________________________
dropout_8 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 65
=================================================================
Total params: 37,889
Trainable params: 37,889
Non-trainable params: 0
```

Figure 5. Model structure and number of parameters of GRU

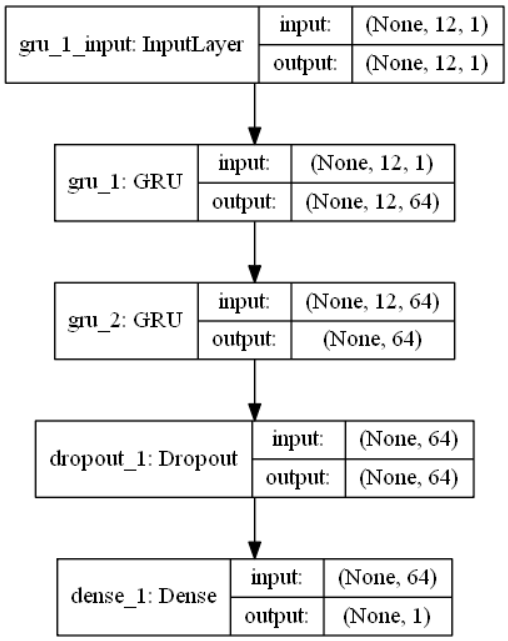


Figure 6. Detailed stricture of GRU

We could see that the first layer has 12 neurons and the second layer has 64 neurons. Then I added a dropout, to random drop 20% neurons in the second layer. Dropout is a technique where randomly selected neurons are ignored during training. They are "dropped-out" randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to the neuron on the backward pass. The last layer is a linear layer with 64 neurons.

In the training process, setting batch size equal to 256, and epochs equal to 150. And after 150 epoch, the validation loss did not change a lot, it is converged.

```
Epoch 1/150
65/65 [==============================] - 2s 31ms/step - loss: 0.0146 - mape: 47.8331 - val_loss: 0.0045 - val_mape: 1
07535.4609
Epoch 2/150
65/65 [==============================] - 2s 24ms/step - loss: 0.0029 - mape: 19.3883 - val_loss: 0.0024 - val_mape: 1
06818.0000
Epoch 3/150
65/65 [==============================] - 1s 23ms/step - loss: 0.0019 - mape: 16.0213 - val_loss: 0.0012 - val_mape: 8
1662.5234
Epoch 4/150
65/65 [==============================] - 2s 23ms/step - loss: 0.0016 - mape: 13.4833 - val_loss: 0.0038 - val_mape: 3
6631.1250
Epoch 5/150
65/65 [==============================] - 1s 23ms/step - loss: 0.0014 - mape: 12.3728 - val_loss: 0.0034 - val_mape: 1
7224.5801
Epoch 6/150
```

Figure 7. GRU training process

After predicting each y, transform the normalized one to the original value.

The following two figures show the predicted trend of one day, April 1st and the whole month of April. We could know the result of this method is really good, with low MSE, for one day prediction and one month prediction.

```
explained_variance_score:0.988559
mape:5.414422%
mae:10.767868
mse:188.997228
rmse:13.747626
r2:0.988274
```

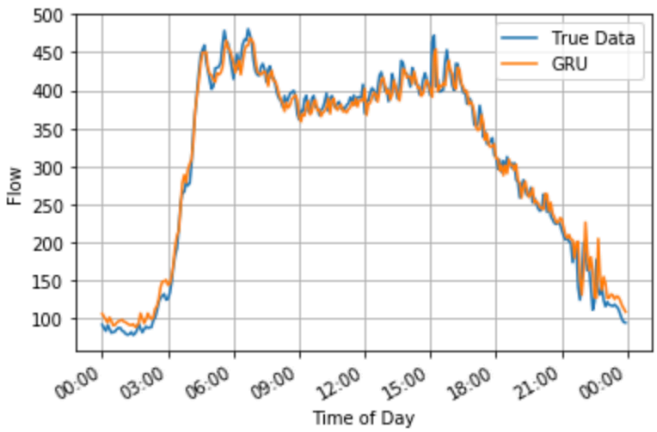


Figure 8. GRU predict trend vs true trend of one day

```
explained_variance_score:0.988559
mape:5.414422%
mae:10.767868
mse:188.997228
rmse:13.747626
r2:0.988274
```

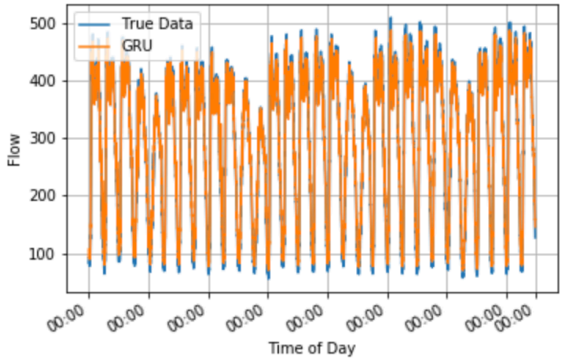


Figure 8. GRU predict trend vs true trend of one month

Part 2:

Different from the method in part 2, here after using the 12 timesteps predict the 13th one, we move a step forward. So the next sample is using the 2th to 12th plus the predicted 13th to predict the 14th.

And it is reasonable that as time goes by, the accuracy would be lower and lower, because there the latter prediction contains all the errors before it. We could see from the following figure.

```
GRU
explained_variance_score:-0.317007
mape:69.736351%
mae:124.764959
mse:21122.935357
rmse:145.337316
r2:-0.324207
```

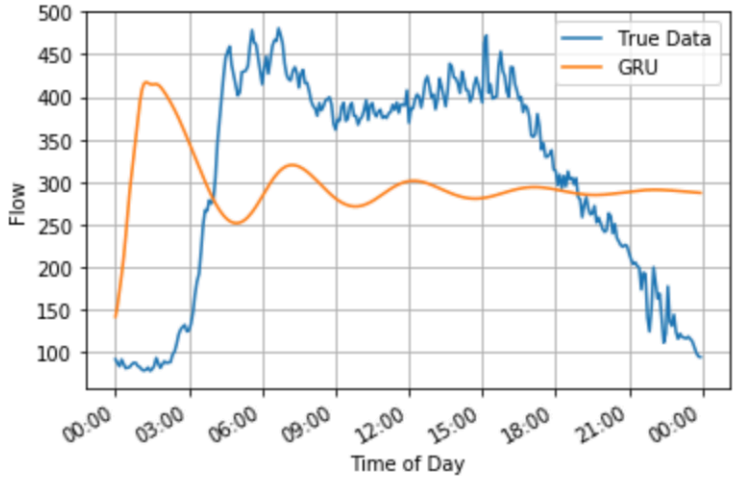


Figure 9. Self recursive prediction of one day

4.5 LSTM for Predicting Time Series Data

Model structure:

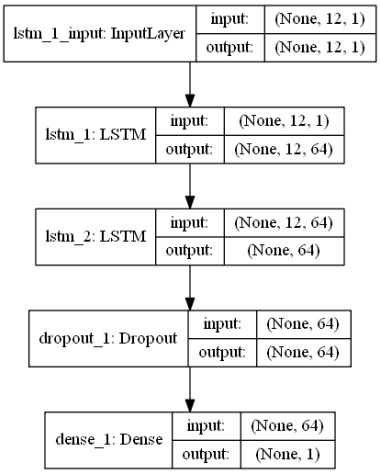


Figure 10. LSTM model structure

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_2 (LSTM)                (None, 12, 64)            16896
_________________________________________________________________
lstm_3 (LSTM)                (None, 64)                33024
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
=================================================================
Total params: 49,985
Trainable params: 49,985
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 11. LSTM model structure with parameters

We could see that the first layer has 12 neurons and the second layer has 64 neurons, then I added a dropout, to random drop 20% neurons in the second layer. The last layer is a linear layer with 64 neurons.

Training process: from the following picture that the validation loss is decreasing continuously and after 150 epoch, it did not decrease a lot, so I stopped training.

```
Epoch 146/150
65/65 [==============================] - 2s 27ms/step - loss: 4.4622e-04 - mape: 5.6199 - val_loss: 5.8960e-04 - va
l_mape: 34850.0820
Epoch 147/150
65/65 [==============================] - 2s 27ms/step - loss: 4.4686e-04 - mape: 5.6316 - val_loss: 5.3937e-04 - va
l_mape: 30662.5176
Epoch 148/150
65/65 [==============================] - 2s 27ms/step - loss: 4.5514e-04 - mape: 5.6239 - val_loss: 7.3605e-04 - va
l_mape: 46978.3359
Epoch 149/150
65/65 [==============================] - 2s 27ms/step - loss: 4.5437e-04 - mape: 5.6348 - val_loss: 6.9261e-04 - va
l_mape: 33275.1133
Epoch 150/150
65/65 [==============================] - 2s 27ms/step - loss: 4.4301e-04 - mape: 5.4090 - val_loss: 7.2311e-04 - va
l_mape: 20679.2812
```

Figure 12. LSTM training process

```
LSTM
explained_variance_score:0.990318
mape:3.958717%
mae:10.793482
mse:210.460841
rmse:14.507269
r2:0.986943
y (288,)
```

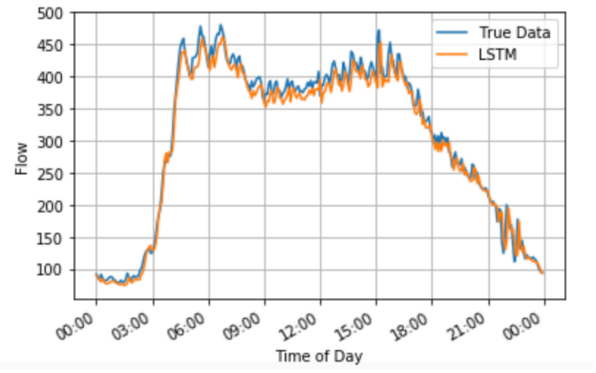


Figure 11. LSTM prediction plot

4.6 Evaluate Model Performance and Model Comparisons

In this part, to fully evaluate each model performance, I made a plot containing all the

prediction plot in order to compare them directly.

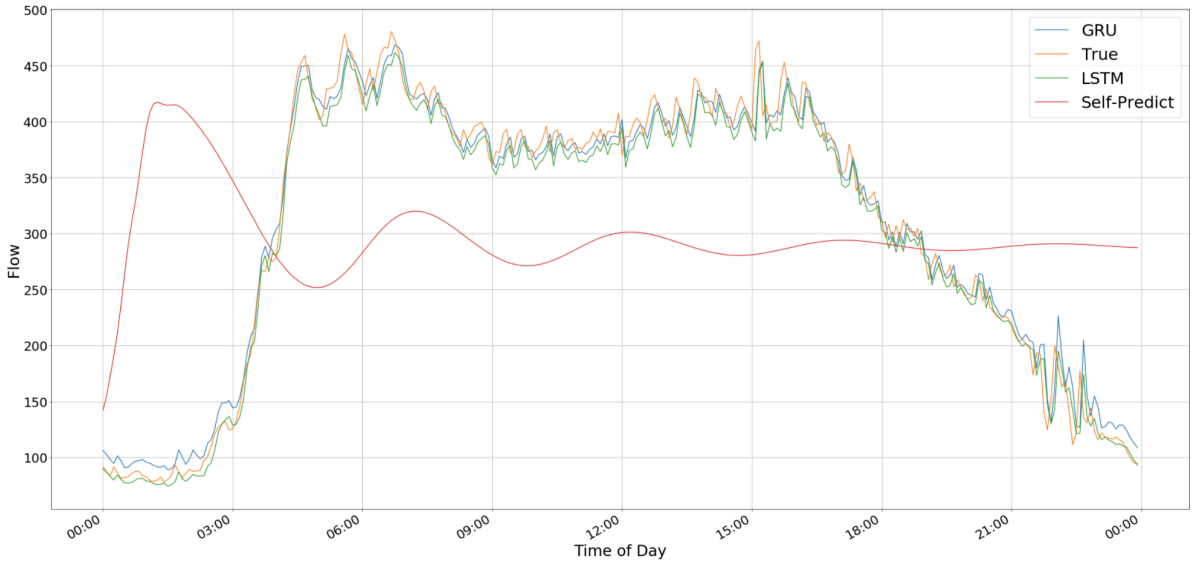


Figure 12. Prediction plots for all plots

Model/Metrics	MAPE	MAE	MSE	RMSE	R2	Explained Variance
GRU	5.414%	10.767	188.997	13.747	0.988	0.988559
Self_Recursive	69.736%	124.765	21122.935	145.337	-0.324	-0.317007
LSTM	3.958%	10.793	210.461	14.507	0.987	0.990318

I calculated several predictive evaluation indicators: RMSE, MSE, MAE, MAPE.

1)Mean Square Error:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

When the predicted value is completely consistent with the true value, it is equal to 0, which is a perfect model; the greater the error, the greater the value.

2) Root Mean Square Error

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$$ It has the same meaning as MSE

3) Mean Absolute Error

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|$$

4) Mean Absolute Percentage Error

$$MAPE = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{\hat{y}_i - y_i}{y_i}\right|$$

We could see from the table above, just as section 4.3 talks about the difference between GRU and LSTM, both of the two perform very well and do not have large difference on the predictive evaluation indicators. Except the MAPE, all the other indicators of GRU model are better than those of LSTM.

For future work, I think I could try more experiments to improve the model paramters,

such as changing the hyper-parameters, to change the lag of 12 less or more, adding the model more layers, like three layers or decreasing the layers to one and then evaluate the model performance.

5. References:

[1] Yisheng Lv, Yanjie Duan, Wenwen Kang, Zhengxi Li, and Fei-Yue Wang, Traffic Flow Prediction With Big Data: A Deep Learning Approach, IEEE Transactions on Intelligent Transportation Systems, 2015, 16(2):865-873
[2] https://github.com/xiaochus/TrafficFlowPrediction
[3] https://www.codecademy.com/articles/normalization
[4]https://towardsdatascience.com/how-and-why-to-standardize-your-data-996926c2c832