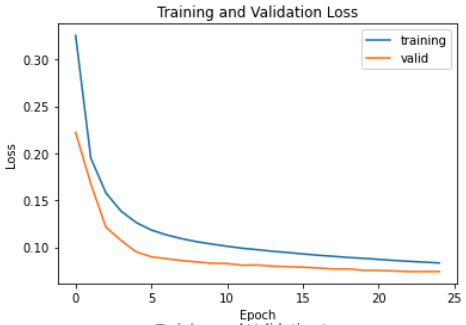
Csc578-"Class Project", Part (A) Time Series Kaggle
Serena Yang
Kaggle Name: **SerenaMing Yang** Ranking: **25**
Score: **607**

The purpose of this final project is to based on building a recurrent neural network to find the best model for predicting the traffic volume for the second hour based on the five hours of input data. I initially did data exploration (EDA), from the exploration, I found that

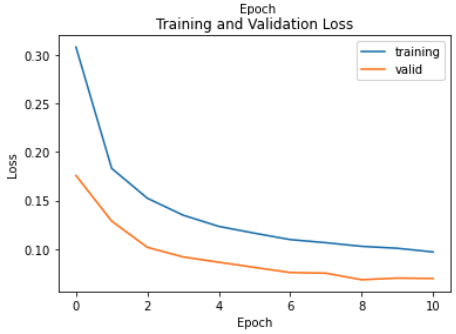
1. For 'holiday column', there are 12 different values. *In the dataset, expect 'None' value：*
   'Labor Day' shows 7 times;
   'Martin Luther King Jr Day','Christmas Day','Thanksgiving Day','New Years Day' shows 6 times;
   'State Fair', 'Memorial Day', 'Indeoendence Day', 'Columbus Day', 'Washingtons Birthday','Veterans Day' shows 5 times.
2. There are 11 unique values in 'weather_main' feature. Most days are 'Clouds' and 'Clear,' and 'Smoke' and 'Squall' barely happen. There are 38 unique values in 'weather_description' feature. Most weather is 'sky is clear,' and a lot of weather descriptions barely happened. There is strong correlation between variables 'weather_main' and 'weather_description'.
3. There are five cycles from 2012 to 2018 and a gap from 2015 to 2016. The temperature is overall decreasing. For the rain_1h and snow_1h have barely happened.

After exploring the dataset, I did some data preparation. First, I converted the 'Date Time' to seconds, daily and yearly periodicity for time series prediction, and use sin and cos to present the time as a point of a day and of a year. Also, I dropped the variable 'weather_description' since this feature is highly correlated to the feature 'weather_main.' Then I splitted the data into 70% as the training dataset, 20% as the validation group, and 5000 records as the test dataset. Moreover, I used the mean and standard deviation of the training dataset to normalize the data. After the data preparation completed, I used the method that the TensorFlow time series tutorial provided to build the data windowing for a 5-hour input window, offset equals to three and label width equals to 1.

After everything settled down, I started with the recurrent neural network baseline trying to predict. The basic recurrent neural network with one layer and one dense has about 0.07 validation loss and 89% r^2, which the performance plot has shown at the right. Even though the RNN baseline model with the very baseline performs pretty well and the final training result and the validation result are pretty close, it still shows some underfitting and the training loss could be lower.

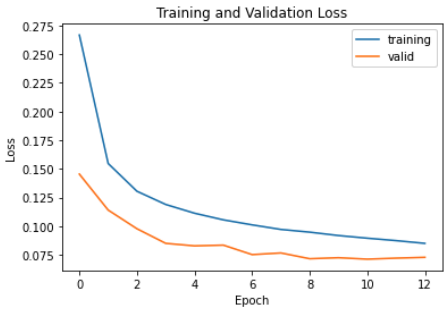


I also tried changing the batch size to 64 and 240, trying to control the training case into the network before the weight got updated, which can help improve the performance. After

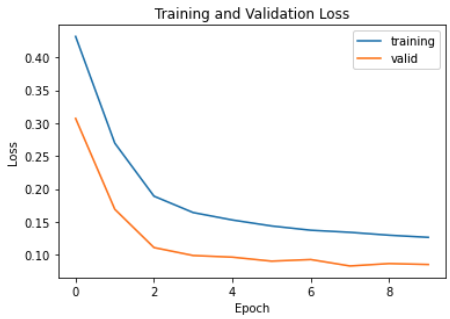


comparing the results, size 64 performance was the best. After trying different batch sizes, I added recurrent dropout = 0.1 and changed the number of epochs. The performance improved slightly with 0.068 validation loss, 0.0525 MAE, and 89% r^2.

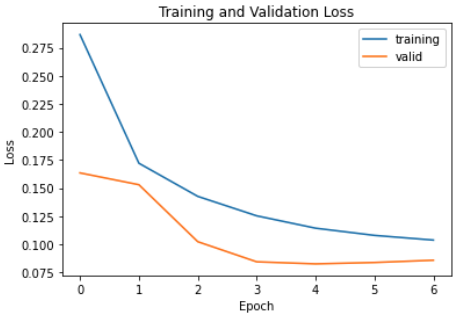
After changing the tuning parameters, I tried bidirectional RNNs, which combined two RNN models with size 64 as the batch sizes, 0.1 dropout, and one more dense layer. One LSTM layer is for moving forward to the sequences dataset, and another RNN model is for moving backward. The mean absolute error is around 0.0571, and the r^2 is about 0.90. I also used this model to predict the result for Kaggle, with about an 806 score.



From the class material, I learned that RNN/LSTM is suitable for dealing with the sequences dataset by treating the data with a 3D shape. CNN is good with digging the spatial correlation from the dataset and treating data as 3D. By combined these two methods, it can help to improve the performance of predicting. After trying the recurrent neural network baseline, I added a 1D convolutional neural network layer into the original model to extract features and used RNN dense layer as the last layer to analyze the features across the consequence time steps. From the performance plot shown on the right, we can find the validation loss is still around 0.08 and 87% r^2.



I also tried a CNN and RNN model with the following hypermeter tuning as shown below; however, the performance was not good, with 0.2 validation mean absolute error and 0.0853 validation loss. I think this model is an interesting one. I thought the performance would be improved when the model became more complicated. After applying this model, I learned that sometimes, a simple model could perform better, but a complicated model will not constantly improve the accuracy. And from the performance plot, we can see that there is still some underfit problem.




```
model = models.Sequential()
# Shape [batch, time, features] => [batch, time, lstm_units]
model.add(layers.LSTM(20, input_shape=(serie_size, n_features), return_sequences=True))
model.add(layers.LSTM(12, activation='relu', return_sequences=True))
model.add(layers.LSTM(2, activation='relu'))
model.add(layers.RepeatVector(serie_size))
model.add(layers.LSTM(serie_size, activation='relu', return_sequences=True))
model.add(layers.TimeDistributed(layers.Dense(units=1)))
model.add(layers.Dropout(0.1))
model.add(layers.Flatten())
# Shape => [batch, time, features]
model.add(layers.Dense(units=1))
```

For this project, I think the difficulty level is medium to hard. Even though there is a tutorial that provides the general solving process, I spent a lot of time doing the model with a score of around 2200 on Kaggle. Then, after I did variable drop from the original dataset, changed the method to normalize the dataset, and turned off the shuffle function, re-follow the instruction from the tutorial; finally, I did the best model with a score of around 806 on Kaggle. Also, from this project, I learned that different variables selected at the data exploration step and making a simple model could affect the final prediction accuracy a lot.