Predicting traffic flow with machine learning

Martijn Noorlander
University of Twente
P.O. Box 217, 7500AE Enschede
The Netherlands
m.s.noorlander@student.utwente.nl

ABSTRACT

This work describes a way of predicting traffic flow with machine learning. To accomplish this goal, several research questions have been created. These are Q1: What machine learning techniques can be used to accurately predict traffic flow? Q2: How can we use the predicted traffic flow to avoid congestion? Q3: What is the influence of non-cooperative vehicles on this model? The first question will be used to select the most accurate technique from an LSTM, GRU or CNN. The technique can be used to create a rerouting mechanism in SUMO to prevent congestion. Finally the work describes ways to evaluate the effect of non-cooperative vehicles on the rerouting mechanism to test the effectiveness on a more realistic scenario.

Keywords

Machine Learning, Traffic Prediction, Congestion Avoidance, SUMO, Keras, Convolutional Neural Network, LSTM, GRU

1. INTRODUCTION

1.1 Background

Autonomous driving is a quickly growing subject in the modern world. With the increasing amount of self-operating vehicles a large number of challenges arise. One of these challenges is avoiding congestion in an increasing amount of traffic. A possible solution for this problem would be to use Machine Learning to predict traffic flow and use the predicted flows to redirect vehicles on a different route. By using the real time location of vehicles, neural networks could be used to predict this flow. Being able to avoid a congestion in a road network yields several benefits, these include decreased travel time and fuel consumption for the cars.

1.2 Previous work

The literature study was first used to identify several ways of using neural networks to predict traffic flow. Convolutional Neural networks (CNN), Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRU) networks have been used quite often for this problem as mentioned in Zhao 2017[7], Ma 2017[6] and Dai 2019[2].

34th Twente Student Conference on IT Jan. 29nd, 2021, Enschede, The Netherlands.

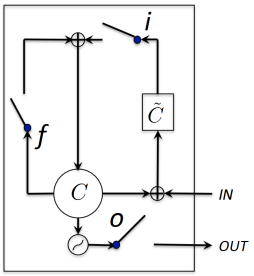

Figure 1. LSTM Cell (taken from Rana 2016 [5]), with Input Gate i, Forget Gate f and Output Gate o

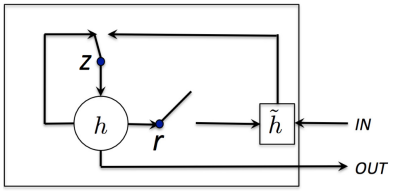

Figure 2. GRU Cell (taken from Rana 2016 [5]), with Update Gate z and Reset Gate r

In Ma 2017 [6] a CNN is proposed that uses traffic as images and then predicts traffic flow and is usable for large-scale transportation networks. By converting the traffic to temporal-spatial images (as shown in Figure 1 in [6]), these are processed with a Convolutional Neural Network.

Zhao 2017 [7] proposes a solution that uses an LSTM to predict short-term traffic. Long Short-Term Memory (LSTM) cells consist of three gates, an input gate, an output gate and a forget gate. A sample cell is shown in Figure 1 and taken from Rana 2016 [5]. This structure allows LSTM cells to remember values over certain time intervals and the cell can use the gates to regulate the information flow through the cell. The structure makes them very suitable for time series prediction. Zhao creates a two-dimensional LSTM network, which allows for temporal-spatial predictions.

Dai 2019 [2] proposes a GRU based solution to predict traffic flows and this is compared to a CNN solution. The main difference between an LSTM cell and a GRU cell is the absence of a third gate. Instead of an input, forget and output-gate, only an update and reset gate exist. A GRU cell is shown in Figure 2 and is taken from Rana 2016 [5]. This structure is accomplished by merging the input and forget gate into a single gate, reducing complexity and increasing speed. Kaiser 2015 [8] provides more information.

All three of the implementations provided promising results, however none of them were directly compared. Com-

paring these three methods on the same data set would therefore provide a good indication on the relative performance.

In Fouladgar 2017 [3] a decentralized deep learning-based method is proposed for detecting the congestion state for a node in the network based on the congestion state of the predecessors. This is particularly interesting since it resides closely to the main research question. However, instead of having a node predict its own congestion state, this specific research focuses on predicting the congestion state from the viewpoint of a car's route and adjusting their route accordingly.

In Code 2018 [1] a realistic traffic scenario for SUMO is described that has been verified with real world data. This provides options for this research to evaluate the congestion predictions by running them through real-world scenarios. To keep the research within a scope, the scenario will have to be stripped from traffic types such as public transport and pedestrians.

1.3 Research Questions

The goal of this research is to avoid traffic congestion by predicting traffic flow with machine learning. One of the problems that arises with this solution is the influence of non-cooperative vehicles, which could disturb the flow and worsen a congestion. This research will focus on the following questions:

- What machine learning techniques can be used to accurately predict traffic flow?
- How can we use the predicted traffic flow to avoid congestion?
- What is the influence of non-cooperative vehicles on this model?

The first question will help us select the most accurate machine learning technique that can be used for the second and third part of the research. The second question will help us evaluate the selected technique in identifying and preventing traffic congestion. The final question will help us evaluate the effects of non-cooperative vehicles in the aforementioned problem.

1.4 Paper Structure

The paper will start by explaining the approach that was used to tackle the problem, this includes the data generation and implementation of the different models. Afterwards the results, conclusion and future work will be discussed.

2. APPROACH

The first step of this research is to compare LSTM, GRU and CNN neural networks on their performance in traffic predictions. These models will be applied to datasets that will be generated with SUMO. SUMO (Simulation of Urban MObility) is an open source traffic simulation package developed by the German Aerspace Center which allows for modeling and simulation of traffic. Multiple scenarios will be generated which will show us the impact of a high or low complexity network. The LSTM, GRU and CNN will be built using Keras in Python, since this will allow us to easily interact with the TraCI library from SUMO. The predictions will be evaluated using the Root mean Squared Error (RMSE).

After comparing the different neural networks on generated scenarios, the most accurate will be chosen for the second and third research questions. From here, the next step will be to use predicted flows to find expected traffic congestion and compare these with simulations of the scenarios in SUMO Using these results, we will update the scenarios to include vehicle route redirection for all vehicles to avoid traffic congestion by using the predicted flows. The scenarios will then be simulated in SUMO to evaluate the results. Congestion will determined by setting a maximum number of cars that are allowed to be on a node before it is declared a congestion.

Finally, to answer the third research question, the scenario will be modified to only redirect a portion of the vehicles to simulate non-cooperative vehicles. Different portions of non-cooperative vehicles will be simulated to compare these results to a fully cooperative scenario.

2.1 Data Generation

To generate the required training data, SUMO was used. A road network of the UT campus was exported from OpenStreetMap and then converted into a SUMO compatible network with the SUMO `netconvert` tool. After generating the network, routes were generated with the `randomtrips.py` tool that is included with SUMO. Finally a simulation was run with SUMO to get traffic data that was exported to a csv. This method was used for all three models, although a smaller portion of the campus was used for the CNN, as explained in the implementation. The map used for the LSTM and GRU was 3km by 2km, the smaller map used for the CNN was 780m by 1000m.

Figure 5 shows the larger SUMO network that was created for the LSTM and GRU, within this network 700 different routes of varying length were created. The simulation span was 3600 seconds with a time step length of one second. The (overlapping) routes are shown in figure 4.

For both the GRU and LSTM models the data was split up into sequences that were captured by creating a rolling windows over each of the cars' locations. The sequences were 20 time steps each, with the steps after that being used as expected output for the model. The then shuffled sequences were split into a training and validation set with a 90% split. All data was normalized to be between `[0,1]` by applying a min-max normalization.

$$\text{minmax}(x) = \frac{x - \min(x)}{\max(x) - \min(x)}$$

As mentioned later, Convolutional Neural Networks rely on images to proess data. Therefore, the CNN model was fed with a data generator that could create frames for a specific batch size. The generator transformed sequences created in a rolling windows into images by merging all locations for a specific timestep into an image. Each car received a unique color to distinguish them between different images. An example of a sequence of these images is shown in figure 3.

2.2 Models

All deep learning models were implemented in Python using Keras with a TensorFlow GPU backend. All models were trained by using the Mean Squared Error (MSE) as loss function, with the `Adam` optimizer. Because training and evaluating models is a time intensive task, not all different configurations could be tested. A desktop with an Intel i7 4770K, 16GB RAM and a GTX1060 was used for all training and validation.

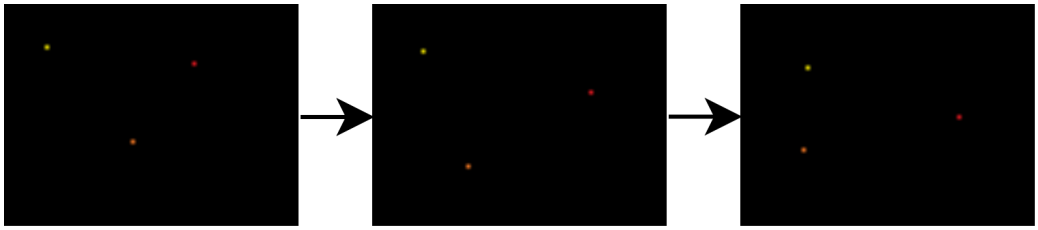

Figure 3. CNN Input sample

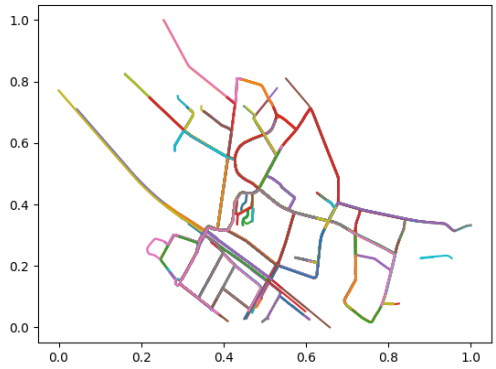

Figure 4. Routes generated for the LSTM and GRU

3. IMPLEMENTATION

3.1 LSTM

The LSTM model was implemented as follows:

Layer Type	Input Shape	Output Shape
LSTM(64)	(steps,2)	(steps)
LSTM(64)	(steps,2)	(steps,2)
LSTM(64)	(steps,2)	(steps,2)
LSTM(64)	(steps,2)	(2)
Dense(2)	(2)	(2)

Four LSTM layers with 64 units were stacked, where the first three had the option `return_sequences=True` enabled. This allows LSTM layers to feed the sequence to the next layer, with the final block having the option disabled so a single timestep is passed to the Dense output layer. Stacking LSTM layers allows for a greater complexity for the representation of the data. The input of the LSTM consisted of a certain amount of steps and then two features, the x and y coordinates of the cars.

3.2 GRU

The GRU model was implemented as follows:

Layer Type	Input Shape	Output Shape
GRU(64)	(steps,2)	(steps,2)
GRU(64)	(steps,2)	(steps,2)
GRU(64)	(steps,2)	(steps,2)
GRU(64)	(steps,2)	(2)
Dense(2)	(2)	(2)

As explained before, a GRU cell is a simplified LSTM cell. Because of this reason, the GRU network was given the same depth and complexity as the LSTM network, with this we can give a clear difference in speed and accuracy between the two models.

3.3 CNN

The CNN model was implemented as follows:

Layer Type	Input Shape	Output Shape
Input	(780, 1000, 2*st)	(780, 1000, 2*st)
Conv2D(2*st)	(780, 1000, 2*st)	(780, 1000, 2*st)
Activation(”relu”)	(780, 1000, 2*st)	(780, 1000, 2*st)
BatchNormalization	(780, 1000, 2*st)	(780, 1000, 2*st)
Dropout(0.2)	(780, 1000, 2*st)	(780, 1000, 2*st)
Conv2D(2*st)	(780, 1000, 2*st)	(780, 1000, 2*st)
Activation(”relu”)	(780, 1000, 2*st)	(780, 1000, 2*st)
BatchNormalization	(780, 1000, 2*st)	(780, 1000, 2*st)
Dropout(0.2)	(780, 1000, 2*st)	(780, 1000, 2*st)
Conv2D(2)	(780, 1000, 2*st)	(780, 1000, 2)

In the table `st` indicated the length of the input sequence. Because convolutional neural network need images as inputs, the input shape has been adjust to the resolution of the network. As can be seen in the input shape of the model, the total amount of data is significantly larger than that of the LSTM and GRU. Because the model is significantly harder to train, I have taken a simplified part of the network. For reference, to use the same network as the LSTM and GRU, images of a resolution of 3000x2000 would have to be used. The model can be divided into two parts, the first consists of stacked blocks of Conv2D, BatchNormalization and Activation layers. Batch Normalization is used to avoid overfitting and to increase the efficiency of learning by normalizing the output of the activation layer. Finally a Dropout layer is applied, also to prevent overfitting. These blocks were stacked to increase the complexity and accuracy of the model. The final layer is used to step from the multitude of channels to only two, the predicted next frame.

3.4 Multistep predictions

All of the aforementioned models output a single prediction step based on a certain amount of input steps (20 for the LSTM and GRU, 3 for the CNN). Predicting multiple steps was done by using the output of the previous prediction step as input for the next. If accurate, the multi step input can be used to calculate the density of cars in all locations of the network.

4. RESULTS

In the next few sections, the results of the different models are analysed and then compared with each other. To evaluate the results, the Root Mean Squared Error (RMSE) was calculated on the prediction and actual values of a data set. The min-max scaling that was applied to the data set was reversed before calculating the values.The root mean squared error gives us the root of the average of squared errors between two data sets.

$$\text{RMSE}(y, \hat{y}) = \sqrt{MSE(y, \hat{y})}.$$

Figure 5. Campus network used for data generation

Table 1. LSTM Validation results

Predicted Steps	RMSE	MAE
1	4.7	3.21
2	6.50	4.63
3	9.86	7.24
4	14.04	10.45
5	18.67	14.01

Table 2. GRU Validation results

Predicted Steps	RMSE	MAE
1	6.98	5.44
2	13.72	11.38
3	25.44	21.52
4	41.44	35.32
5	61.94	52.94

$$\mathrm{MSE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} (y_i - \hat{y}_i)^2.$$

Table 3. CNN Validation results

Predicted Steps	RMSE (Pixel to Pixel)
1	0.84*

$$\mathrm{MAE}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} |y_i - \hat{y}_i| .$$

The Root Mean Squared Error for the LSTM and GRU was calculated by taking the root of the `sklearn.metrics`[4] `mean_squared_error` function. To calculate the MAE (Mean Absolute Error) the `sklear.metrics` `mean_absolute_error` function was used. For calculating the RMSE of the CNN the `skimage.metrics` `mean_squared_error` was used, this function compares two images pixel wise and calculates the error between the colors of all pixels.

4.1 LSTM

The LSTM was trained for 150 epochs on a set of 52k sequences consisting of 20 time steps each, a batch size of 1000 was used. During training, the time per epoch was 1 second. With this, the results can be found in table 1. Because the RMSE and MAE have the same scale as the dataset, the results are in meters.

4.2 GRU

The GRU was trained for 150 epochs on a set of 52k sequences consisting of 20 time steps each, a batch size of 1000 was used. During training, the time per epoch was 1 second. With this, the results per amount of predicted steps can be found in table 2. Because the RMSE and MAE have the same scale as the dataset, the results are in meters.

4.3 CNN

The CNN was trained for 150 epochs on a set of 900 sequences, consisting of 3 time steps each. The training time per epoch was 66 seconds, with 7 sequences as batch size. Increasing the batch size would result in the GPU running out of memory, switching to training with the CPU (to increase available memory) would increase the time per epoch to about 9 minutes per epoch. With this configuration the results can be found in table 3.

It is important to note that the RMSE of the CNN and GRU/LSTM are not directly comparable since they are based on different datasets. For the CNN the RMSE is calculated on the expected and predicted images, these include all locations that do not contain cars and thus heavily skew the average squared error, since they will always be zero. This means that the RMSE does not give us an error in terms of distance between two positions, but an error in terms of the color of the pixel not matching to that of the expected image. To compensate for this, an attempt was made to extract the predicted car locations from the predicted images and calculate the RMSE in a similair way to the LSTM and GRU. However, due to the inaccuracy of the CNN the colors of the car locations did not match to those of the expected output and often cars were missing. Because of these large errors it was not possible to create an accurate RMSE based on location errors for the CNN. For the same reason it was chosen not to predict more than a single timestep.

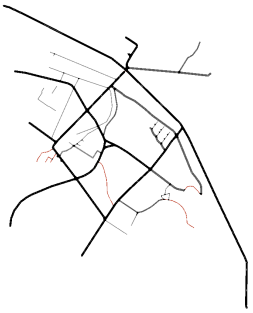

Figure 6. Simplified network used for the CNN

4.4 Analysis of results

Both the LSTM and GRU are significantly faster to train than the CNN, even with the reduced network size for CNN. One of the major reasons is that for a specific time step and car location, an entire frame of dimension (780, 1000, 2) needs to be processed. On the other hand, both the LSTM and GRU need a frame with dimension (1, 2) to represent a single car on a certain timestep. When multiple cars are driving in the frame that is passed to the CNN, the efficiency of the data in the frame is slightly increased, but will still not be close to that of the LSTM or GRU.

Training the LSTM and GRU was very close in time (both reported 1 second per epoch during training), even though the GRU should offer better performance. One cause could be the complexity (or lack thereof for the still relatively small road network of the campus) of the models, which would need to be increased for real world usage on larger road networks. This could possibly lead to a clear advantage in speed for the GRU.

The accuracy of the CNN was quite poor. The RMSE of the CNN was calculated on the difference between the predicted and actual dataframe of the model on a per pixel basis. As mentioned earlier, this means that a majority of the data represents locations where no car will drive and the value of those pixels will always be 0 and therefore skew metrics such as RMSE. Another reason for the decreased accuracy is the small dataset used for training to keep training times reasonable. The training history shows that the model has difficulty with correctly fitting, because the validation data loss is oscillating quite heavily. This can be seen in figure 7.

As expected, the LSTM was more accurate than the GRU. Especially when predicting multiple timesteps ahead, the LSTM showed a clear accuracy advantage over the GRU.

5. CONCLUSION

Based on these results, the LSTM provides us with the most accurate traffic flow predictions. After the LSTM, the GRU follows as a second best option in terms of accuracy. The current configuration of the CNN provides the worst results and is currently unusable with regards to the missing cars in the predictions.
When looking at complexity the LSTM and GRU provide a clear edge over the CNN. Both the preprocessing and training are significantly faster, since the data sets are more efficiently organized. The CNN is very slow in both preprocessing and training, because the used data structure was very inefficient. Based on the current results, we can therefore conclude that an LSTM model is currently the best answer for our first research question.

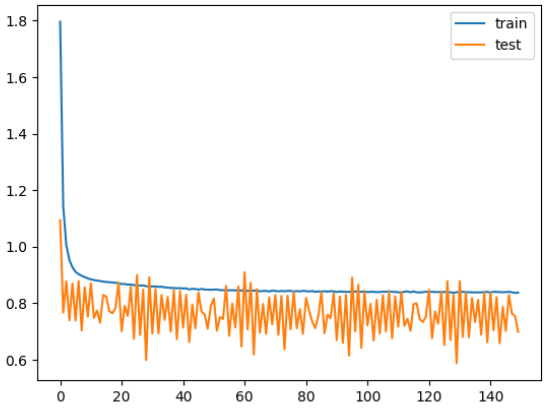

Figure 7. CNN training history, train represents the RMSE loss for the training dataset, test represents the RMSE loss for the validation dataset

6. FUTURE WORK

In this section two topics are discussed, improvements to the prediction models and work on the congestion avoidance.

6.1 Improving the models

One of the major issues of the CNN is the amount of data that was necessary to train the network. One way to solve this would be to create a better representation of the road network in the model. Currently non-drivable locations are included because the frames are the size of the map. By mapping only roads and other driveable locations to positions in the frame, the efficiency would be hugely increased. This would result in faster training times and possibly more accurate results since it would be easier to train on a larger dataset. This would however decrease the flexibility of the system, where currently only previous drives of cars need to be known to learn the network, the network would need to be known before training the model.

For both the LSTM and the GRU more different model configurations could be tested. During this work only a small amount of different configurations was empirically tested. Improving the models and increasing the training dataset will probably result in a higher accuracy.

Finally, to make the models more suitable for real world use, a larger road network would need to be simulated and trained.

6.2 Congestion avoidance

To answer the second research question, "How can we use the predicted traffic flow to avoid congestion", a significant more accurate model is needed. If found, several ways could be used to determine whether there is a congestion state. A simple, but inefficient method would be to iterate over the network and calculate the density of vehicles on locations in the network. For large networks, this method would be unusable. An interesting research topic, would be to use a Convolutional Neural Network to classify traffic congestion. The second part of answering this research question will reside in running simulations that divert different amounts of cars based on the detected congestion. Statistical analysis can then be used to determine whether the prediction and redirection actually has

the desired effect. Data from these simulations can also be used to answer the third research question.

7. REFERENCES

[1] L. Codeca and J. Härri. Monaco SUMO Traffic (MoST) Scenario: A 3D Mobility Scenario for Cooperative ITS. In *SUMO 2018, SUMO User Conference, Simulating Autonomous and Intermodal Transport Systems, May 14-16, 2018, Berlin, Germany*, Berlin, GERMANY, 05 2018.

[2] X. X. Guowen Dai, Changxi Ma. Short-term traffic flow prediction methodfor urban road sections based onspace–time analysis and gru. 2019.

[3] R. E. Mohammadhani Fouladgar, Mostafa Parchami and A. Ghaderi. Scalable deep traffic flow neural networks forurban traffic congestion prediction. 2017.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[5] R. Rana. Gated recurrent unit (GRU) for emotion classification from noisy speech. *CoRR*, abs/1612.07778, 2016.

[6] Z. H. J. M. Y. W. Y. W. Xiaolei Ma, Zhuang Dai. Learning traffic as images: A deep convolutionalneural network for large-scale transportationnetwork speed prediction. 2019.

[7] X. W. P. C. Y. C. J. L. Zheng Zhao, Weihai Chen. Lstm network: a deep learning approach forshort-term traffic forecast. 2017.

[8] Łukasz Kaiser and I. Sutskever. Neural gpus learn algorithms, 2016.