Taxi Driver Trajectory Classification using Siamese Neural Networks

Jannik Haas
WPI Data Science
Worcester Polytechnic University
100 Institute Road, Worcester MA, 01609
jbhaas@wpi.edu

1 INTRODUCTION

Sequences or trajectories are part of our everyday life and in big cities, taxi trajectories make up a large part of that. Being able to analyze these trajectories to distinguish one driver from another or other tasks can be done using different Neural Network structures. In this paper we look at five days worth of trajectories each from five hundred taxi drivers to develop a model that will be able to classify any two trajectories as belonging to the same driver or different drivers.

2 PROPOSAL

In many classification problems we have few classes and many data points for each, however, often we have many classes and few data points for each. This means that the traditional multi class classification methods may not work well since there is not enough data to learn the differentiating features between that many classes. Few shot and meta learning are established to be able to handle these types of problems.
We will use a Siamese Neural network structure consisting of RNN layers that takes in two trajectories and outputs a prediction of whether the trajectories came from the same driver or two different drivers. The Siamese network structure consists of one shared neural network- in our case a RNN with some fully connected layers- which is shared by both trajectories. This means that we are only training one RNN with the same weights for two trajectories. This RNN outputs a set of features extracted from the trajectories which is then passed into a distance layer which examines how close these two trajectories are to then output a single dimension with a sigmoid activation layer to determine if they were from the same driver or not.

3 METHODOLOGY

3.1 DATA PRE-PROCESSING

Plate	Longitude	Latitude	Time	Status	Timestamp
The plate of the taxi driver (in our case 0-500)	The longitude of the location	The latitude of the location	The number of seconds since midnight	0 - empty/seeking passenger 1 - carrying passenger	The timestamp of the record

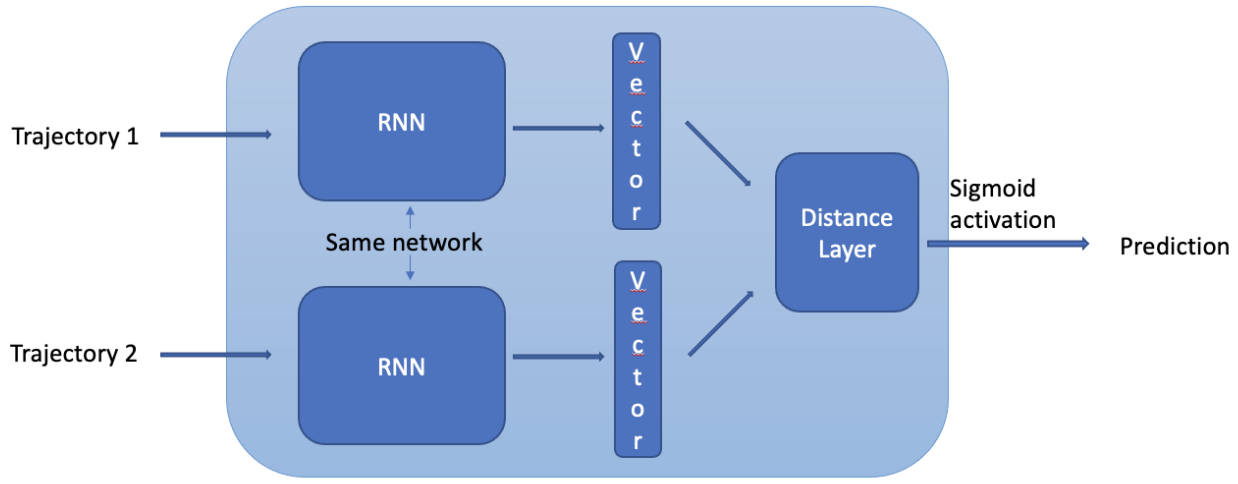
The original data was in a dictionary format where the keys were the 500 driver IDs and the values were 5 trajectories for each driver. For the data preprocessing I first created pairs of trajectories to train the model on. For each trajectory of each driver I selected both a random trajectory from that same driver as well as one random trajectory from a random different driver. This meant that instead of 2500 training points I now had 5000 training points. If the two trajectories were from the same driver then the label was 1 and if they were different drivers the label was 0.
The trajectories varied greatly in length and a short trajectory could be matched up with a much longer. For this I tested both cutting the longer trajectory short to match the shorter one as well as repeating the shorter one to match the length of the longer one. Ultimately the results didn't vary much, however, cutting the longer trajectory shorter gave better results and resulted in faster training time.
Since the trajectories are very long, training an RNN on such long trajectories is not only time consuming, but it's very difficult to extract a small feature space to represent such a long trajectory. For this reason I cut each pair of trajectories into a set of pairs of sub trajectories. Instead of feeding this set of sub trajectories into the model, I trained the model on just one pair of sub trajectories. This increased my training dataset tremendously and reduced training time. For the pairing of the sub trajectories I had two strategies. First I kept sub trajectories at the same time steps paired together i.e. the 1st sub trajectory of trajectory 1 would be paired with the 1st sub trajectory of trajectory 2, the 2nd with the 2nd and so on. This had the advantage of keeping the time consistent in sub trajectory pairs as driver trends most likely vary throughout the day, we would be comparing the beginning of one drivers' shift with the beginning of the other drivers' shift and so on. On the other hand since the trajectories were not the same length, this meant the end of one drivers' shift would most likely be compared with the middle of another drivers'. I also shuffled the sub trajectories so that each one was randomly paired with another sub trajectory from the other trajectory. My goal was to enhance the models' robustness by training on a more challenging problem. This also allowed me to reasonably extend one trajectory to match the longer trajectory, rather than cutting a trajectory short. This in turn also created more training data. Overall the best model was trained with the approach of not shuffling and keeping the time steps paired together, however, in the future I would like to do a direct comparison keeping all the other variables the same and only changing the shuffling parameter.

3.2 FEATURE GENERATION

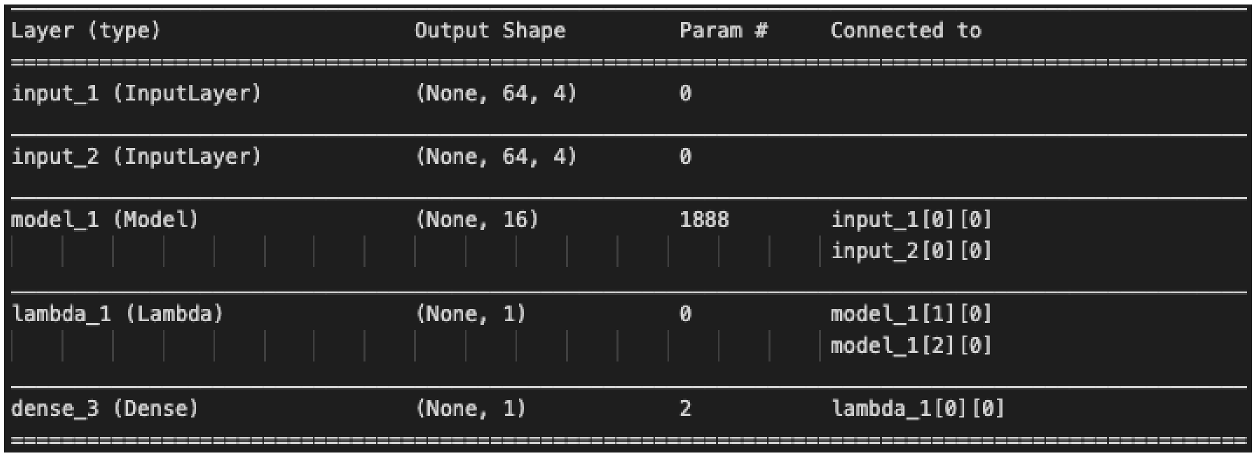
There was no need for feature generation as the model performed very well with just the data presented. Each trajectory that was fed into the model consisted of a set of points of [Longitude, Latitude, Seconds since midnight, Status].

3.3 NETWORK STRUCTURE

Siamese Network Structure



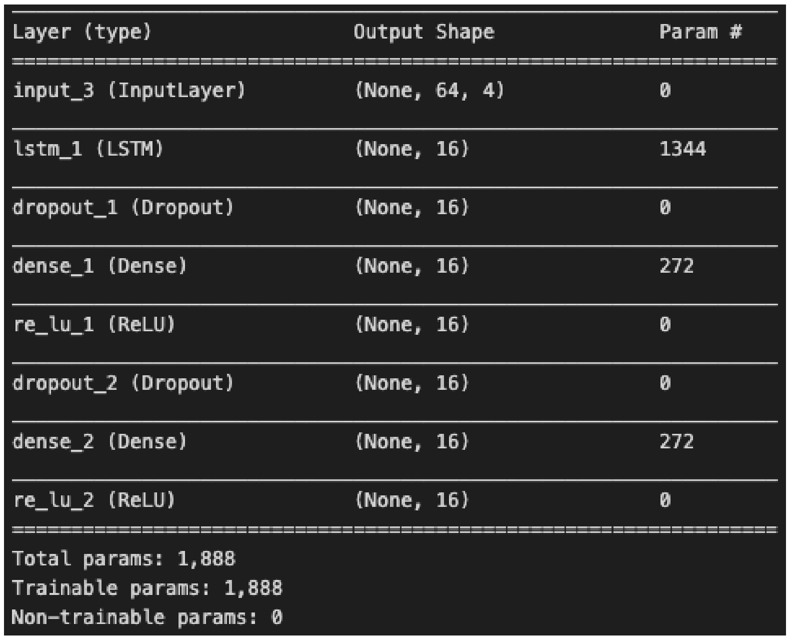
The Siamese model structure is outlined above where two trajectories are each fed into the same RNN network to produce a feature vector. These two feature vectors are then passed into a distance layer to measure their similarities to then come out with a prediction.
I focused my efforts on the RNN model in the above figure and kept the remaining model the same with the exception of testing two distance functions for the distance layer. The complete network structure can be seen below, where the model_1 is the RNN model.



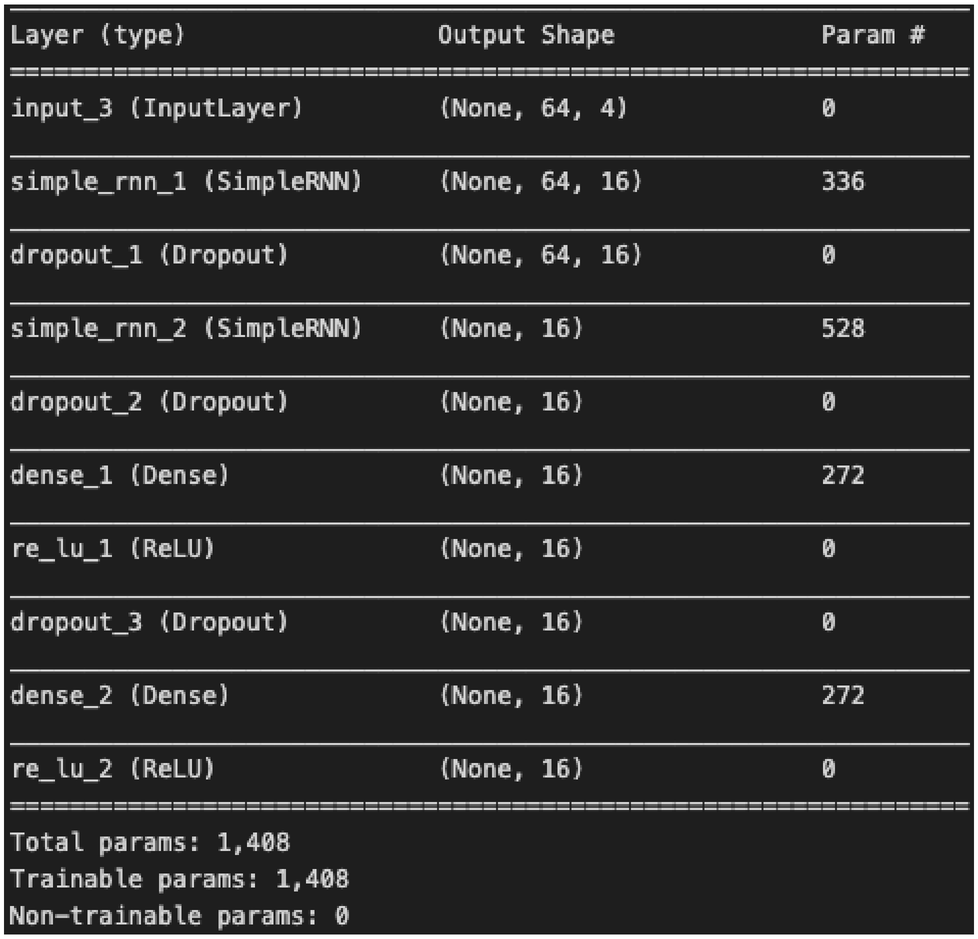
```
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 64, 4)        0
__________________________________________________________________________________________________
input_2 (InputLayer)            (None, 64, 4)        0
__________________________________________________________________________________________________
model_1 (Model)                 (None, 16)           1888        input_1[0][0]
                                                                 input_2[0][0]
__________________________________________________________________________________________________
lambda_1 (Lambda)               (None, 1)            0           model_1[1][0]
                                                                 model_1[2][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 1)            2           lambda_1[0][0]
==================================================================================================
```

RNN Structure

The two RNN structures I experimented with were using LSTM layers and SimpleRNN layers, which fed into two fully connected dense layers. For the best LSTM model a single LSTM layer with 16 hidden units was used. The structure can be seen below:


```
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         (None, 64, 4)             0
_________________________________________________________________
lstm_1 (LSTM)                (None, 16)                1344
_________________________________________________________________
dropout_1 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_1 (Dense)              (None, 16)                272
_________________________________________________________________
re_lu_1 (ReLU)               (None, 16)                0
_________________________________________________________________
dropout_2 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_2 (Dense)              (None, 16)                272
_________________________________________________________________
re_lu_2 (ReLU)               (None, 16)                0
=================================================================
Total params: 1,888
Trainable params: 1,888
Non-trainable params: 0
```

The SimpleRNN layers are simpler and therefore I experimented with using multiple layers of different sizes, however the best performing model used the following structure:


```
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         (None, 64, 4)             0
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, 64, 16)            336
_________________________________________________________________
dropout_1 (Dropout)          (None, 64, 16)            0
_________________________________________________________________
simple_rnn_2 (SimpleRNN)     (None, 16)                528
_________________________________________________________________
dropout_2 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_1 (Dense)              (None, 16)                272
_________________________________________________________________
re_lu_1 (ReLU)               (None, 16)                0
_________________________________________________________________
dropout_3 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_2 (Dense)              (None, 16)                272
_________________________________________________________________
re_lu_2 (ReLU)               (None, 16)                0
=================================================================
Total params: 1,408
Trainable params: 1,408
Non-trainable params: 0
```

Distance Layer
For the distance layer I used the euclidean distance between the two feature vectors for a similarity score. I experimented with using the manhattan distance, since the nature of the problem is taxi trajectories it only seemed appropriate, however, the euclidean distance gave better results overall.

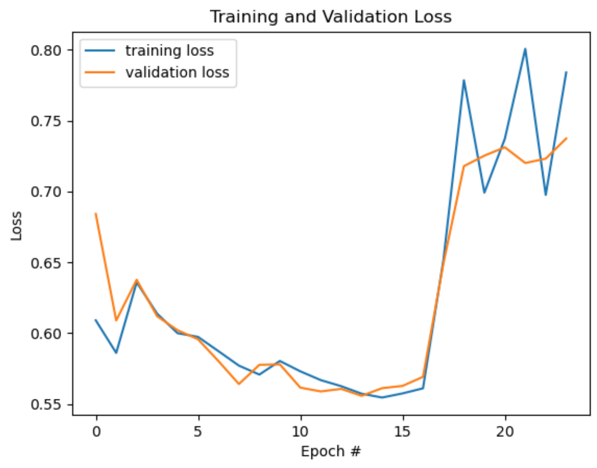
3.4 TRAINING AND VALIDATION PROCESS

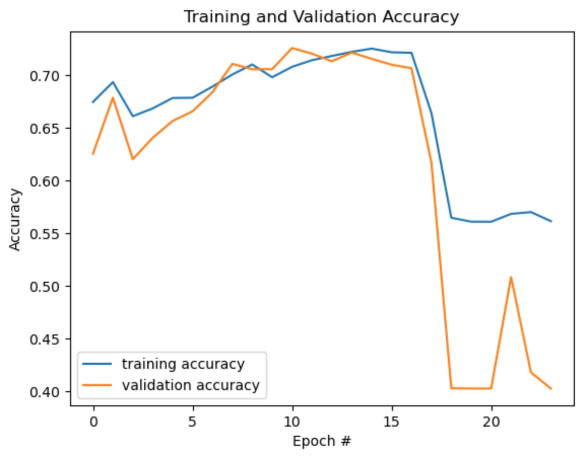
For the training of the model, the keras built in fit() function was used with batch training. The model was compiled using binary cross entropy loss since the nature of the problem is binary classification and the optimizer used was the Adam optimizers. The best performing models were able to converge around 20 epochs.
Since the model was only trained on a pair of small sub trajectories from each pair of trajectories, I expected the performance to be slightly worse than the real validation. For the real validation I took the predictions of each pair of sub trajectories and simply took the majority vote (Full Credit to Mason/Jingyang for the idea from their Project 2 results). This increased the final validation score by about 5% accuracy for almost all the models I tested.

4 RESULTS

4.1 TRAINING AND VALIDATION RESULTS

Below are the loss plot and accuracy plots for both the testing and validation for the LSTM model. As you can see the loss decreases quickly and steadily for the first 15 epochs but then suddenly spikes up. The accuracy follows an inverse trend where it steadily increases for both training and validation and then suddenly drops even below 50% which is worse than a random guessing model. The models were checkpointed however and the best performing model was saved at epoch 15.





The RNN model showed a similar trend where the accuracy increased quickly and then plateaued and later dropped slightly. For the RNN model the accuracy for both testing and validation decreased around epoch 32.

4.2 MODEL COMPARISONS

The LSTM model took much longer to train per epoch due to its complexity, however it did also converge faster than the RNN model. Overall they showed similar final performance on the validation set with the LSTM model beating out the RNN model by a couple percentage points. The simple RNN required two layers to learn the same complexity as a single LSTM layer of the same size. I experimented with using more SimpleRNN layers, however I was unable to obtain any better results.

4.3 HYPERPARAMETER TUNING

I started with a learning rate of $1.5e^{-4}$, however I found that the loss did not decrease enough so I increased the learning rate to $1.5e^{-3}$. This produced steady loss over epochs, however, as we saw from the training plots above, it may have been too small and caused sporadic changes toward the end of training. I also had many training iterations where the model immediately got stuck at a local maxima and was stuck at about 40% accuracy without learning anything. I think the optimal learning rate for this problem lies somewhere in between the $1.5e^{-3}$ and $1.5e^{-4}$.
Batch size was set to 32 and produced great results. Traditionally the lower batch sizes work well for these types of problems so no tuning was done here.
The sub trajectory lengths were set between 16 and 128 with 64 producing the best results. This was a trade off since the final model would take the mode prediction of all the sub trajectories of the trajectories so a small sub trajectory would give us more classifiers per classification, however, a longer sub trajectory may contain more information to discern trends between drivers. Ultimately 64 produced the best results as it was the best balance.

5 CONCLUSION

The LSTM and RNN models were both able to produce good results with accuracies around 80%, with the LSTM model just beating the RNN model. Overall I was surprised at how good the model was able to extract features from just the simple trajectories themselves without any sort of feature engineering in the preprocessing stage. Potentially the simplicity of the trajectories allowed the model to more freely extract features from the raw data, rather than doing feature engineering ahead of time and extracting data.

REFERENCES

"Simple. Flexible. Powerful." *Keras*, keras.io/.

Rosebrock, Adrian. "Siamese Networks with Keras, TensorFlow, and Deep Learning." *PyImageSearch*, 17 Apr. 2021, www.pyimagesearch.com/2020/11/30/siamese-networks-with-keras-tensorflow-and-deep-learnin g/.