COS30018: Intelligent Systems

Project Assignment – Option A (Topic 3)

Traffic Prediction Model

Jack Richardson

Samuel Hauser

Jake Whitfield

Harrison Feldman

[Introduction 3](#_Toc1170611639)

[Overall system architecture 3](#_Toc2036135389)

[Data Preparation for Model 3](#_Toc1017515006)

[SCATS Site Data Processing 4](#_Toc1385636381)

[Training Methods 4](#_Toc1292700857)

[Testing Results 6](#_Toc62702077)

[Route Calculations 8](#_Toc1352937290)

[GUI – Harrison 9](#_Toc1315655923)

[Implemented search/optimization techniques 9](#_Toc1223417382)

[Scenarios/examples to demonstrate how the system works 10](#_Toc1114908006)

[Critical analysis of the implementation 10](#_Toc190672920)

[Summary 10](#_Toc652907849)

[References 10](#_Toc1649230534)

[Images 11](#_Toc2063769508)

[Who Did What? 11](#_Toc425427788)

[Jack Richardson 11](#_Toc929481431)

[Sam Hauser 11](#_Toc618222204)

[Jake Whitfield 12](#_Toc294095924)

[Harrison Feldman 12](#_Toc661195842)

# Introduction

This project, put simply, converts an existing GitHub repository of the original TrafficFlowPrediction model by user ‘xiaochus’, to work with a Boroondara traffic flow dataset and, provide a user with multiple travel routes via road, between two points, identifying them from fastest to slowest.

# Overall system architecture

## Data Preparation for Model

To get the Boroondara dataset working with the existing model, the Pandas library was used to restructure the data in a way that the model could interpret for training and testing.

Before passing the dataset through to the model, a new data-frame object is created from Pandas as a structure for the Boroondara dataset. As each row of the original dataset contained an entire day of 15 minutes increments for traffic flow in that particular scats site, each row had to be pivoted so that each 15 minutes interval of traffic flow data had its own row. The result of pivoting the data is an exponential increase in rows, compared to the original dataset.

Table

Description automatically generated*Figure 1.1 - Data-frame before cleaning*

Graphical user interface, table

Description automatically generated with medium confidence

*Figure 1.2 - Data-frame after cleaning*

As demonstrated in figures 1.1 and 1.2 above, the original Boroondara dataset provided 4194 rows of data with each row representing every 15-minute interval that day. After a pivot was performed on the dataset, 4194 rows multiplied into 402,337 rows, giving us a reasonably sized dataset for creating an 80/20 split for training and testing.

## SCATS Site Data Processing

In preparation for turning each SCATS Site Number into a node for searching for a route, the data had to be first formatted so that the SCATS Number, Longitue, Latitude and Street names could be retrieved. The process\_node() function first creates a dataframe, removes duplicate rows according to the ‘Location’ column and returns the last non-zero longitude and latitude values for the designated SCAT site. This was to prevent any bad entries into the dataset such as the 0 values for longitude and latitude for SCAT Site 4266.

Another dataframe is then loaded to get a list of the street names and directions leading out of the specified SCAT Site. This data is then cleaned to have a list containing just the street name and the direction rather than the additional street being intercepted.

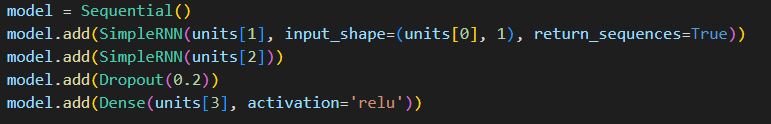
Using the clean data for the street names, they are then iterated through in order to find the closest node also on the road. By removing rows from a new dataframe that do not contain the street names, plus another step of removing the rows from the specified scat site, the remaining rows are then compared to find the SCAT with the closest longitude and latitude values to that of the SCAT Site. If a value can be found, the street name, direction of travel and connected SCAT Site are then appended to a list.

Finally, the data is collated and returned with the SCATS Site Number, Longitude, Latitude and a list of connections detailing the Street Name, Direction of Travel and Connected SCATS Number. This data is then used to create a series of Nodes for use within the search algorithm.

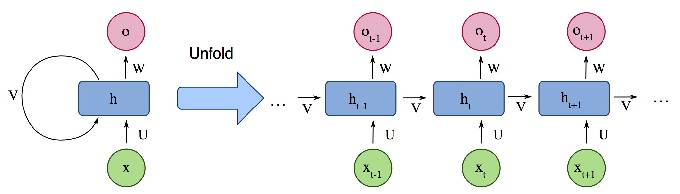
## Training Methods

Two additional neural networks were created for training that used layers which are not used in the three neural networks provided by the original repository. All layers that were used for training are sourced from the Keras’ API within their recurrent layers.

The first custom neural network used for training included Keras’ SimpleRNN layer which was sourced from recurrent layers in the API. SimpleRNN implements the basics of recurrency in a neural network, where the outputs are fed back into the network's own inputs. The activation function used for the SimpleRNN is ‘relu’ this means that all negative values become 0 but, all positive values will remain the same.

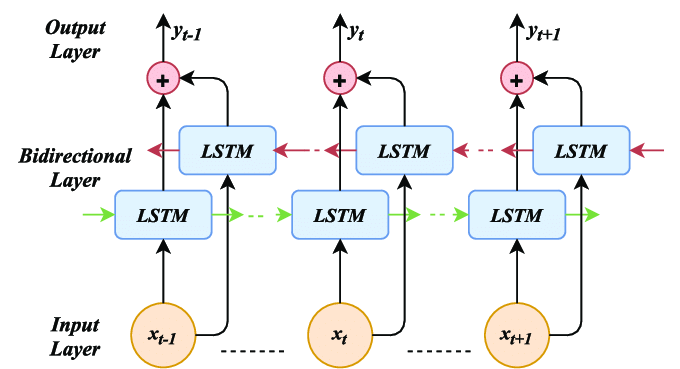


*Figure 1.3 - Plotted results of the 970 scats site for all training methods*



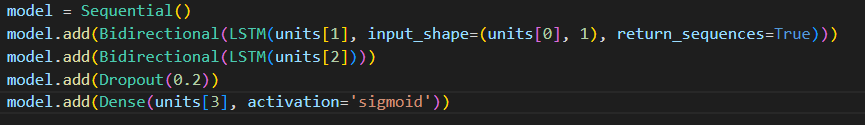
*Figure 1.4 - Architecture of a Simple RNN*

The bi-directional LSTM implemented is an extension of the existing, traditional LSTM used for training the models and, is used in the second custom neural network for training the dataset. The bi-directional, in theory, can improve the model's performance on sequence classification problems, therefore, be a better alternative over the traditional LSTM. While a traditional LSTM only encodes the sequence in a forward direction, a bi-directional encodes in both directions as demonstrated in figure 1.3 below. Once both forward and backward models are trained, they are then merged into a single model.



*Figure 1.5 - Architecture of a Bi-Directional LSTM*

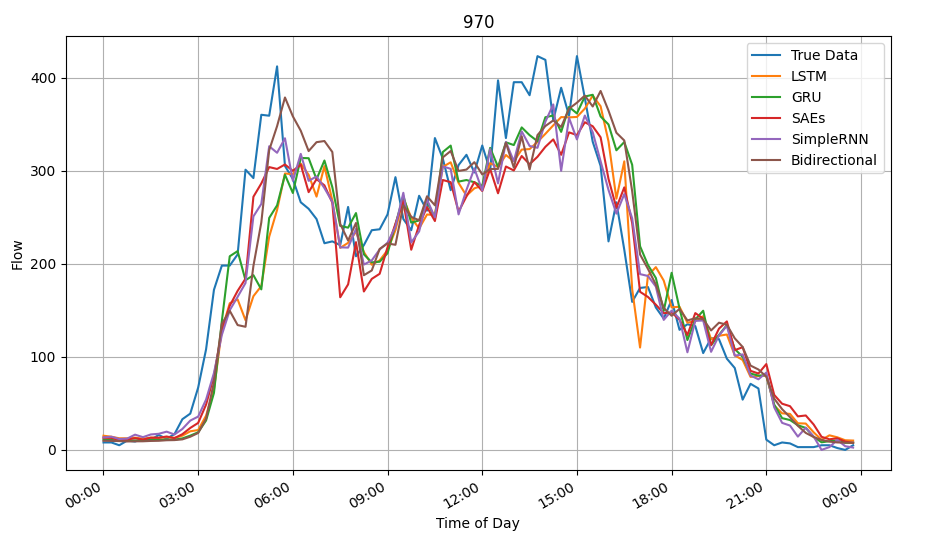
The code block below demonstrates our implementation of bi-directional LSTM. Much like the traditional LSTM included in the original repository, all layers are the same except, when running Bidirectional on the LSTM layer, a forward and backward train are generated and merged. The activation function used in the following block of code is ‘sigmoid’ this means all negative values become 0 and, all positive values greater than 1, become 1, while the remaining values between 0 and 1 are untouched.



*Figure 1.6 - Code block of Bi-Directional LSTM*

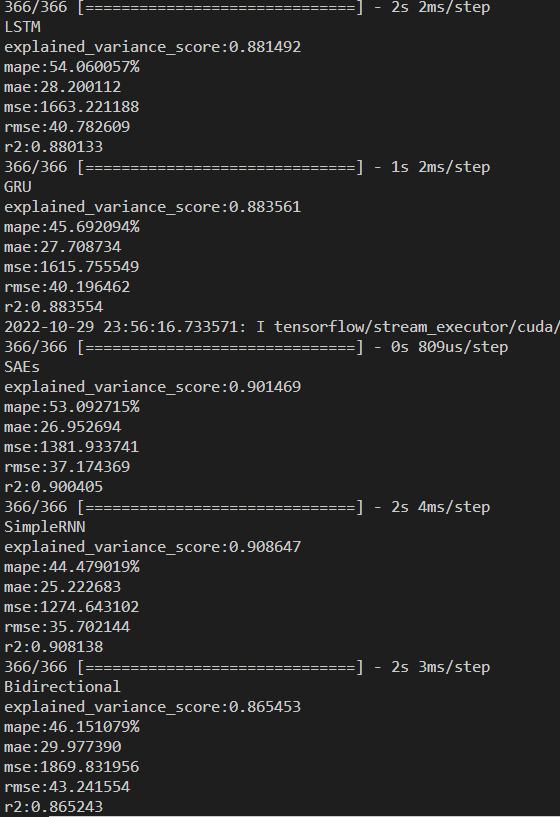
## Testing Results

The model is tested by charting the predicted values for each method against one another with the true data from that scats site across the course of the day, the given example is done off 3 epochs, and as shown from a relatively low level of epochs a consistent trend is shown with SimpleRNN leading the most accurate method.



*Figure 1.7 - Plotted results of the 970 scats site for all training methods*

The results depicted in figure 1.7 of the 970 scats site identify that, where the flow would consistently trend upwards, the models would tend to slightly underestimate the true flow data during that time period. When the flow would consistently trend downwards over time, all models would tend to slightly overestimate the flow for that time period.



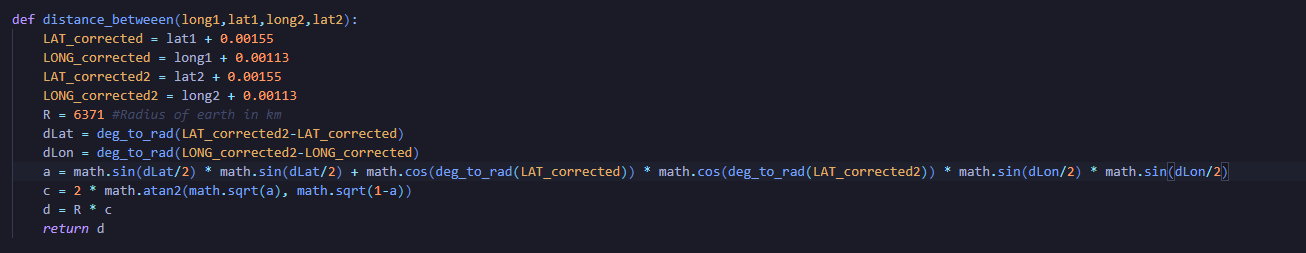
*Figure 1.8 - Metric results for the testing of all implemented models*

The mean average percentage error (MAPE) is the metric used to calculate the efficiency of a model. MAPE is calculated by averaging all the absolute percentage errors from each forecasted value therefore the model with the lowest MAPE would be considered the most accurate. When running main.py to call the test methods, the metrics are printed for the user to make an evaluation on the model. The test results in figure 1.8 show that SimpleRNN scored the lowest MAPE, meaning its predictions fell closest to the true data, making it the best suited neural network which trained scats site 970.

As expected, the Bi-directional LSTM, outperformed the traditional LSTM by a MAPE value of 7.91. However, both neural networks had the lowest efficiency out of the five networks.

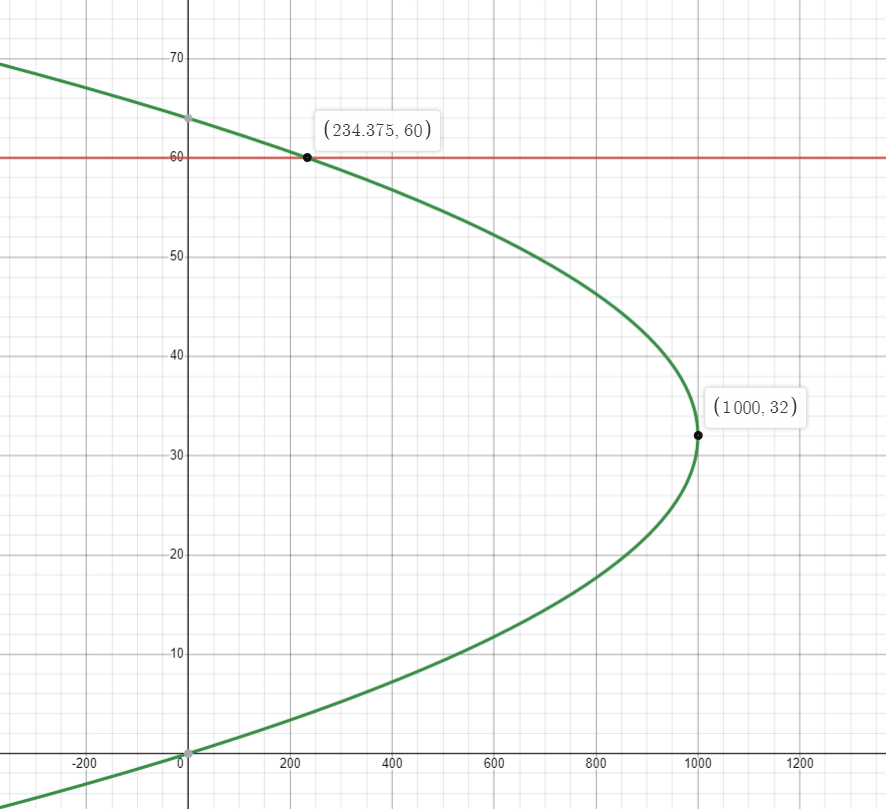
## Route Calculations

Some functions are used to assist the model in estimating the time to get between two SCATS sites, firstly a distance between function is used to calculate the distance (in kilometers) from one set of coordinates to another



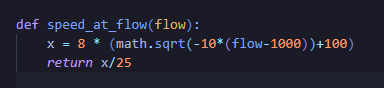
*Figure 1.9 – Distance between function*

In addition to the distance function, using the fundamental diagram of traffic flow, adjusted to meet this project by Matthew Coulter and his group, can be used to determine the speed with respect to the current flow.



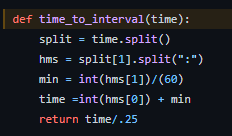
*Figure 1.10 – Fundamental diagram of traffic flow*

Programmatically the function looks like this. Where it is made into an equation, where you solve the equation for Y(speed) given X(flow).



*Figure 1.11 – Speed at flow function*

It is necessary to determine the interval out of 96 that the current time is so that it can be used to feed into other supplementary functions, hence the time\_to\_interval function was created



*Figure 1.12 – Time to interval function*

It takes a time in the format of ‘2006-06-01 13:30:00’ and separates and calculates the decimal time and divides by ¼ to gain that section.

These functions are used together to calculate the time it will take to get between two specified SCATS sites with the predicted traffic flow from the model, the calculations are done for adjacent SCATs sites and will have to be used repeatedly to get the expected travel time for a route.

## GUI – Harrison

# 

# Implemented search/optimization techniques

The search algorithm used to determine the fastest roads the user should take between the starting and ending points is A-Star. The A-Star search is an informed search algorithm, where the search path is determined by a heuristic which estimates the cost to the goal. ‘A Star’s’ heuristic must be admissible, meaning it never overestimates the cost to reach the goal so, when the algorithm checks each node, it will add the cost of the distance travelled, with the nodes estimated distance to the goal node. When the algorithm traverses the tree, it will store each node in a priority queue, only expanding the nodes with the smallest distance. Once it reaches the goal, if there are other nodes with a shorter distance in the priority queue, it will expand them until the goal node has the shortest distance of all paths, from there, recursion can be called to retrieve the rest of the nodes in that path.

In the case of route navigation for this project, up to five routes need to be calculated for the user to choose from. Once the fastest route has been discovered from the targeted area of roads. It will be pruned from the tree so that the next time the algorithm is run, it will not select the same route and, the next best route will be calculated using the same algorithm. Fastest routes will be stored and returned to the GUI.

# Critical analysis of the implementation

The current implementation of the traffic flow modelling bases calculations on the flow at a SCAT site instead of modelling the flow between the SCAT sites. This was due to limitations in our ability to implement the full model and conduct any testing on a simpler prototype due to ongoing difficulties. As a result, in the current build of the project, the direction of traffic within a SCAT site has not been considered in the model.

All aspects of the system work in isolation but full integration has not been completed at time of submission, leaving a little more to be desired.

# Summary

Overall, we were able to get all systems functioning independently except the GUI. For model training, the SimpleRNN produced the most accurate results when forecasting the flow. Bi-directional LSTM proved to be more accurate than its parent LSTM neural network but, overall was not a viable option for this dataset as SimpleRNN, SAE’s and GRU all proved to be more accurate by producing a lower MAPE score.

The A-Star algorithm was correctly implemented and returned successful validation tests in the .NET framework but, within the final week before submission, we ran out of time while trying to get the node object to communicate with the search after porting the A-Star to Python.

Node objects for the search successfully queried a model respective to its scats site and return the flow between itself and a target scats site. The value returned was to be used as the cost when determining the shortest path within the search function.

# References

* <https://towardsdatascience.com/lstm-and-bidirectional-lstm-for-regression-4fddf910c655>
* <https://en.wikipedia.org/wiki/Fundamental_diagram_of_traffic_flow>

## Images

* <https://www.researchgate.net/figure/Architecture-of-a-simple-recurrent-neural-network_fig3_338644066>
* <https://www.researchgate.net/profile/Augustine_Nwajana/publication/344554659/figure/download/fig3/AS:944635081940995@1602229962458/Bidirectional-LSTM-model-showing-the-input-and-output-layers-The-red-arrows-represent.png>

# Who Did What?

## Jack Richardson

**Code**

* Model
  + Added both new neural networks (SimpleRNN and Bi-directional LSTM)
  + Wrote code to query a scats model for the flow at a particular time.
* Path generation
  + Wrote A-Star search algorithm
  + Wrote method for GUI to call when requesting a list of shortest paths generated by search.
  + Wrote Node class in classes.py
  + Wrote methods to unpack nodes into the path travelled
  + Wrote method to get the total estimated time from start to finish along a particular route.
* Data preparation
  + Cleaned and ran pivot on Boroondara dataset to create new data frame for training the models.

**Documentation**

* Wrote data preparation for model section
* Wrote training methods section
* Wrote training results section
* Wrote search optimisation section
* Wrote Summary
* Voice recordings for data cleaning, model implementation and results

## Sam Hauser

**Code**

* Main
  + Converted charts to plot the correct data compared to original repository
  + Sorted prediction arrays
  + Time to interval conversions
  + Supplementary functions
* Search
  + Cost functions
  + Fundamental diagram conversion to functions.

**Documentation**

* Wrote route calculations section
* Added to Critical analysis
* Edited video & submission of project

## Jake Whitfield

**Code**

* Main
  + Created function top initialise all the nodes for the map
  + Helped with debugging numerous functions
  + Contributed to implementation of retrieving model based on method and SCAT site
* Data
  + Created the process node function to retrieve all the required data for a SCAT site including the number, longitude and latitude and the connections between SCAT sites
  + Added to the process data to create and analyse models for each individual SCAT site
* Search
  + Debugging and conversion of system to utilise Node class functions
* Classes
  + Helped implement Node class for use within the search
* Test Mapping
  + Created basic program for testing and demonstrating the connections between SCAT sites

**Documentation**

* SCATS Site Data Processing
* Added to Critical Analysis

## Harrison Feldman

**Code**

**Documentation**