UPPSALA
UNIVERSITET

Real Time Forecasting of Bus Arrivals Using GPS data

Simon Bethdavid, Simon Blomberg, Simon Hellman

Project in Computational Science: Report

January 2020

PROJECT REPORT

Institutionen för informationsteknologi

1 Introduction

Everyone has experienced a bus not arriving on time while waiting at the bus stop. Sometimes it leaves early, sometimes it arrives late. This sparks frustration, and for some people a willingness to pursue a project in bus arrival forecasting. The problem of accuratlely predicting when a bus will arrive at a bus stop or five stops ahead is a complex one. There are numerous variables determining the outcome—such as weather, bus driver, distance, neighbourhood population, traffic conditions—just to mention a few. In Uppsala County the public transportation is provided by Upplands Lokaltrafik AB (UL). For bus arrival forecasts UL currently uses a third-party tool built on top of a proprietary algorithm.

During the Spring of 2019 there was a group of students at Uppsala university that conducted their Bachelor's thesis [1], *Are we there yet?*, for UL. They implemented a neural network prediction model using the inputs: *time travelled since the start of journey*, *time of day*, *current bus stop* and *final bus stop*. The output was the time travelled between the specified stops. This approach managed to outperform the current forecasting system used by UL [1]. Also during 2019 a master's thesis was written together with Östgötatrafiken AB (ÖT) and their third-party forecasting provider. A type of recurrent neural network called long short-term memory (LSTM) was used. The results from this study were even more promising [2]. Wei Fan and Zegeye Gurmu studied a similar traffic related prediction problem and found that neural networks are well suited for problems in this domain [3].

This project is done in collaboration with UL, with the intention of producing bus arrival forecasts that can compete with the previous works and the currently implemented system. Compared to [1] we will use real time GPS data in order to investigate whether more detailed data will give higher performance for the arrival forecasts. In part, this data will be provided by a partner group from the course *Project in Embedded Systems*. They will create a real time position tracker and a web platform. Their project, together with the forecasting provided by this project will form a proof-of-concept product that intends to show how everything could look to the end user.

2 Architecture

UL saves GPS data from their busses every second and this data has the potential to be used for training a machine learning model to predict bus arrivals. This project uses the GPS data from buses on line 5 in Uppsala going from Stenhagen to Sävja. The goal is to train a machine learning model with this data and have it serve requests for a standalone web server. The real time data used for predictions is gathered from on-board GPS units assembled by our partner group in the course *Project in Embedded Systems*. The data from these units are uploaded to an SQL database on the prediction machine every second. The partner group where also responsible for creating a website displaying the buses together with the predictions from the prediction server (PS). Communication between different parts of the architecture are all handled through HTTP web

requests making it very separable. The PS for instance can be run on a separate machine or even in a cloud environment. The same is true of the SQL database and the website. Figure 1 illustrates the relationships between the different components.

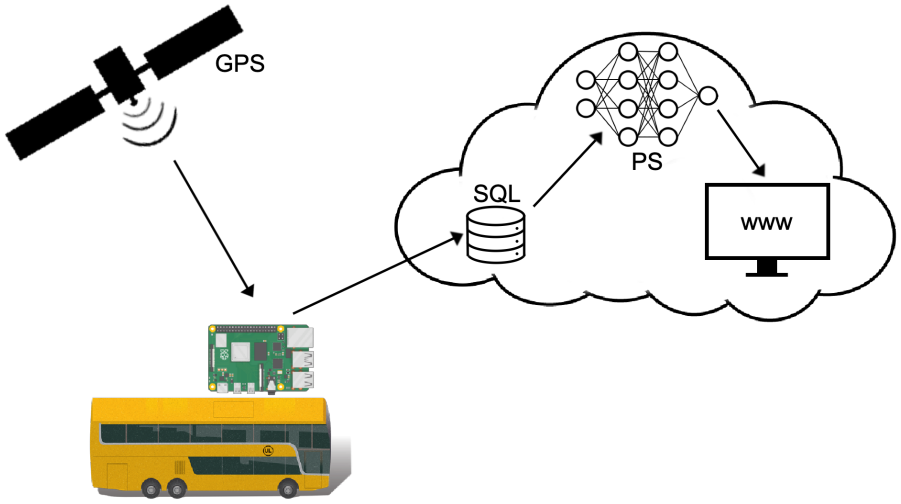

Figure 1: All parts of the system architecture except for the on-board unit are separable and can be executed in a cloud environment (Raspberry Pi image from [4]).

2.1 Prediction Model

This project focuses on optimising a feed forward Neural Network (NN) as previous work point towards a promising result when forecasting transportation arrivals using NNs [3].

In *Neural Networks for Pattern Recognition*, Bishop [5] covers how NNs are implemented. This project uses a feed forward NN with two hidden linear layers. All layers except the last use ReLU activation functions. Batch normalisation was introduced between the linear layers.

For reference, a linear regression model and random forest model were trained with the same input parameters. In the remainder of the report these are referred to as baseline models.

3 Neural Network

NNs was originally invented to simulate the neurons of a brain, now they are used in a wide variety of machine learning tasks to model non-linear relationships.

NNs work by applying consecutive linear transformations and non-linear activation functions to an input vector. The linear transformations are implemented as

matrix multiplications, where each element of the matrix is a optimised parameter. A problem with the linear transformations is the fact that a composition of them is also linear, hence such a network can only model linear relationships. To remedy this we introduce non-linearities in the form of activation functions between the linear layers. A common choice of activation function is the rectified linear unit function (ReLU),

$$f(x) = \max(0, x). \tag{1}$$

B. Hanin and M. Sellke [6] show that a NN with ReLU activations can approximate any continous function with arbitrarily precision. It also has good computational properties which makes it suitable for real time predictions (see, e.g., [7]).

3.1 Loss Functions

A NN is parameterised by its weight matrices. These matrices are chosen to minimise the so-called *loss function*. The loss function is problem specific and is a function of the network prediction and the ground truth. Two typical loss functions for regression problems is the mean absolute error (MAE) loss

$$l(p, t) = \overline{|(p - t)|} \tag{2}$$

and the mean squared error (MSE) loss

$$l(p, t) = \overline{(p - t)^2}, \tag{3}$$

where p is the predicted value and t is the ground truth. $\overline{x}$ represents taking the mean value of x.

This project evaluates four different loss functions,

1. MSE loss,
2. MAE loss,
3. Asymmetric MAE loss,
4. Truncated loss.

The latter two are both visualised in Figure 2. The asymmetric MAE loss is defined as

$$l(t, p) = \overline{\alpha d H(d) - \beta d H(-d)}, \tag{4}$$

where $H(x)$ is the Heaviside function, $d = t - p$, α and β are hyperparameters. By varying the hyperparameters we can penalise errors of a certain sign more than the other. This is relevant since early predictions are worse than late in the buss arrival prediction domain. This loss has the effect of biasing the predictions towards either later or earlier depending on α and β. A result of this is that it cannot be evaluated by its MAE since it will always be higher than for the standard MAE loss, the resulting model trained with this loss function is instead evaluated qualitatively. The truncated loss is defined as

$$l(t, p) = \overline{(-d + \tau)H(-d + \tau) + (d - \tau)H(-d - \tau)}. \tag{5}$$

This loss function has a non-penalised region around zero. Outside of this region the function behaves as a standard MAE. The rationale behind this is that a bus arriving within τ seconds is not a problem that needs to be optimised for. The hope is that this makes the model focus on making the really off-target predictions better, rather then optimising the already accurate predictions.

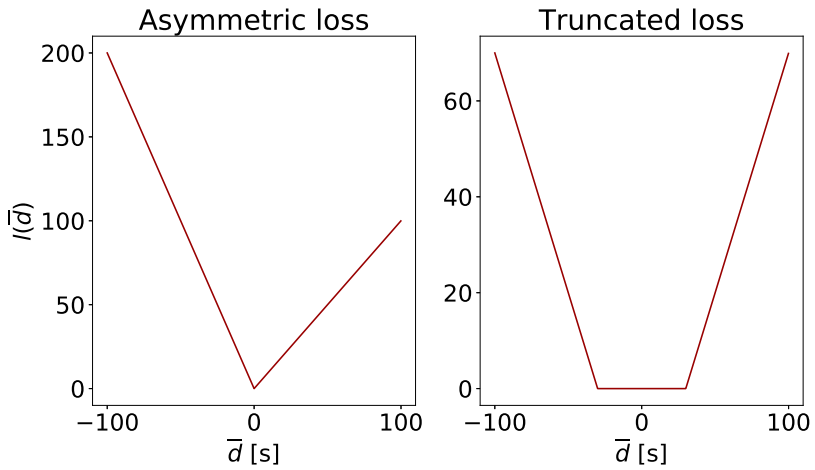

Figure 2: Visualisation of the two custom loss functions — asymmetric loss and truncated loss.

3.2 Optimisation

The weights are updated by iteratively minimising the loss function using a gradient method such as stochastic gradient descent (SGD). It is therefore important that the loss function is locally differentiable with respect to the weights.

This project uses the Adam optimisation algorithm to optimise the loss function. It is a first order optimisation algorithm extended from SGD. Adam was chosen since it has proven to work well optimising stochastic problems of this nature (see in, [8]).

3.3 Batch Normalisation

Due to their high model complexity, NNs are inherently prone to overfitting. The standard solution to this is to introduce a regularisation method, called dropout. Dropout works by randomly ignoring inputs into layers during training to prevent overfitting to specific inputs or samples. A drawback of this method is that it slows down training due to ignoring some features of the input data. Another regularisation option that instead improves training time is batch normalisation. This works by normalising the inputs for each batch of samples [9]. An additional effect of batch normalisation is that it makes the loss function smoother, allowing the use of higher learning rates which in turn speeds up the training.

4 Implementations

The main tools used to implement the prediction server are:

- Python 3.7.1
- NumPy 1.17.3
- Pandas 0.25.3
- PyTorch 1.2.0
- SciPy 1.3.1
- Flask 1.1.1
- ScikitLearn 0.22

4.1 Input Parameters

The model uses a number of auxiliary input parameters along with the GPS data. They are used in order to capture other phenomena that influences how long it will take for a bus to travel a specific distance.

Time since start of trip: All timestamps are calculated since the start of the trip. Probably the most intuitive parameter since the entire project revolves around forecasting arrival times.

Time of day: This parameter intends to capture the different traffic conditions during different times of the day. If one could access the data on the traffic conditions itself, that would be preferred. Having access to that data would also enable one to deal with unlikely disturbances that occur during otherwise calm hours of the day. Since we do not have access to that kind of data we use *time of day* as a proxy for the traffic conditions.

Travel time of previous buses: Another possible proxy for local traffic conditions is the travel time of previous buses on the same route. Compared to other features this one is not in real time, which means that these features will make the model less sensitive to the current state. We hope to improve the overall performance of the model using this parameter. It is also a way of making the model more stable and not too sensitive to the current situation.

Precipitation: This feature consists of the amount rain or snowfall as a single parameter measured in millimetres. It is a reasonable guess that heavy rain or snowfall might affect the traffic conditions. Rain might increase demand and snowfall can affect road passability.

This data is acquired using the Swedish Meteorological and Hydrological Institutes' (SMHI) API [10], specifically data from the weather station "Uppsala Flygplats".

Temperature: Like with rain and snowfall, temperature might have some impact as well. One guess is that lower temperatures could increase demand. The data is acquired in the same way as precipitation, using the SMHI API [10].

4.2 GPS Data Preprocessing

Using raw longitude and latitude data as input features would presumably not do much for the network, unlike humans it cannot correlate the two scalars to a position in the real world. The GPS data needs to be transformed into a representation relating to our problem. A promising metric would be the Euclidean distance to the final stop, but since it is known that the bus keeps to its line, a better representation is the distance along the line towards the bus stop incorporating this new information.

The GPS training data obtained from UL contains a number of irrelevant columns, hence data preprocessing is essential before feeding it to the NN. Cleaning the data results in files containing the columns, timestamp, longitude, latitude and buss index. As the forecasting of bus arrivals is fixed to line 5 in one direction, Stenhagen - Sävja, the next step is to filter out that specific line. This is done by finding all trips that pass by both the start and end stops, extracting the corresponding bus index and finally selecting all data points corresponding to that bus index. This project received 16 days (all within one month) worth of GPS data and after cleaning it resulted in approximately 1000 line 5, Stenhagen - Sävja, trips.

The next step is to convert the GPS data into trip progress. This is done by projecting the GPS coordinates onto a bus line acquired from an API provided by UL [11].

The algorithm to obtain the trip progress is divided into two parts, the first part determines which segment the data point should be projected onto by choosing the closest one, see Figure 3. In reality the segments are smaller and there is only one segment to each data point. The second part is to calculate how far on the segment the bus has travelled. The total travel length is then calculated by summing the previous segment lengths and how far the bus has travelled on the closest segment.

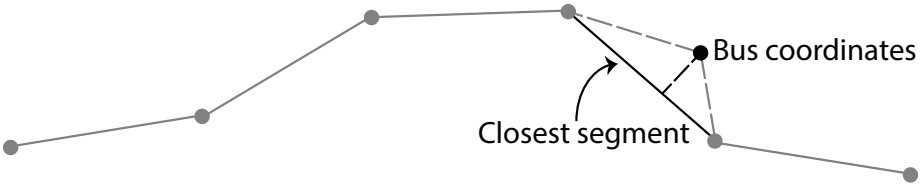

Figure 3: Illustration of segmented bus line.

When calculating the distances to the segments there are three possible cases: A, B and C as shown in Figure 4.

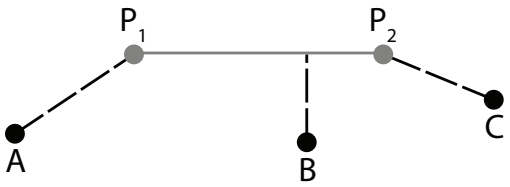

Figure 4: Figure showing the different cases when calculating the distance on the bus line.

For case A and C the distance is the Euclidean distance between A or C and the corresponding endpoint of the segment. For case B, a projection onto the line is calculated via a scalar product as illustrated in Figure 5. This gives both the shortest distance to the line segment, and the relative distance travelled on the segment as denoted by x in the Figure. The variable x is given by,

$$x = \vec{V}_{P_1 B} \cdot \widehat{V}_{P_1 P_2} \tag{6}$$

and the distance by Pythagorean's theorem as

$$d = \sqrt{|\vec{V}_{P_1 B}|^2 - x^2}. \tag{7}$$

Here $\vec{V}_{QR}$ is the vector from point Q to R, $|\vec{V}|$ denotes the length of $\vec{V}$ and $\widehat{V} = \vec{V}/|\vec{V}|$.

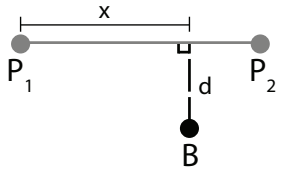

Figure 5: Figure showing the distance needed to get the trip progress.

In Figure 6 two buses are visualises by their trip progress. One can clearly see that the largest disturbance compared to a linear relationship occurs when passing the central station where the buses seem to stand still for a longer period of time.

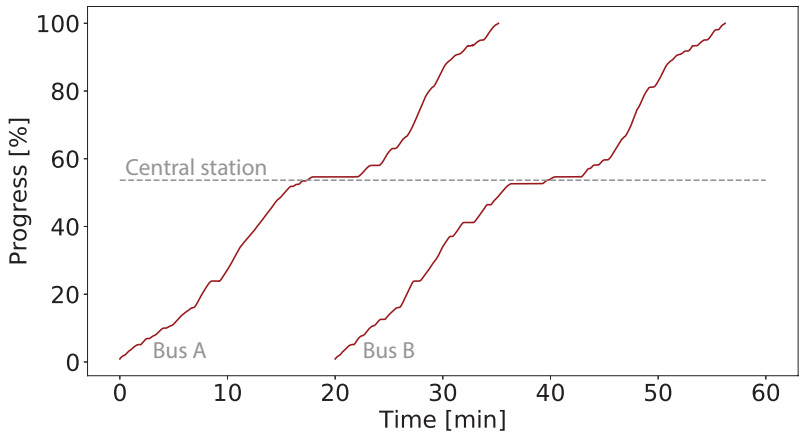

Figure 6: Figure showing the final GPS embedding, progress over time for two buses.

4.3 Sample Generation

To simulate real time requests we generate the input samples to the NN by randomly sampling a trip, choosing a target bus stop according to a triangular distribution and choosing a random start position on the selected trip before the retrieved bus stop. The reason for not sampling the bus stops from a uniform distribution is that it would yield a mean sample distance of about one forth of the total line length. Since we want our model to focus on longer, more difficult predictions, we need the mean distance to be longer and therefore use a triangular distribution which yields a mean distance of one third.

The generated GPS data is then combined with the auxiliary data such as weather, fetched from the SMHI API [10], and time to form the final sample.

Since the model is meant to predict arrival time using real time data collected from GPS units located on the buses, there is a need to quickly create real time samples. Overall this process is very similar to the sample generation process described above. However the travel time of previous buses are not as straight forward to generate, since we need to make sure to collect the underlying data continuously. We solve this by a data collection script that stores historic data in the SQL database. The samples are then generated by combining this data with the real time GPS position obtained from the on-board unit, the arrival location and the auxiliary data.

5 Prediction Results and Discussion

The data is split into three datasets:

- Train — Used only for training the network and optimising the network weights. (72.5 %)
- Validation — Used for evaluating the performance of the network and tuning hyperparameters. (12.5 %)
- Test — Only used for evaluating the final model. (15 %)

The overall model performance compared to the baseline models are presented in Table 1. The NN performs slightly better than both the random forest and linear regression models using the same input features. It is interesting to note that the linear regression model outperforms previous NN results obtained without using GPS data in terms of mean absolute error (72.4 s) (see [1]).

Table 1: Performance of the neural network compared to the baseline models. When evaluating the accuracy a correct prediction is defined to be within 60 s of the actual arrival time.

	Neural Network	Random Forest	Linear Regression
MAE [s]	63.2	69.1	70.4
Accuracy [%]	59.8	53.7	54.9

Figure 7 presents the absolute error compared to the length of the predicted trip. It points towards a fairly linear relationship between the error and trip length. For longer trips the model is more likely to miss-predict than for shorter ones.

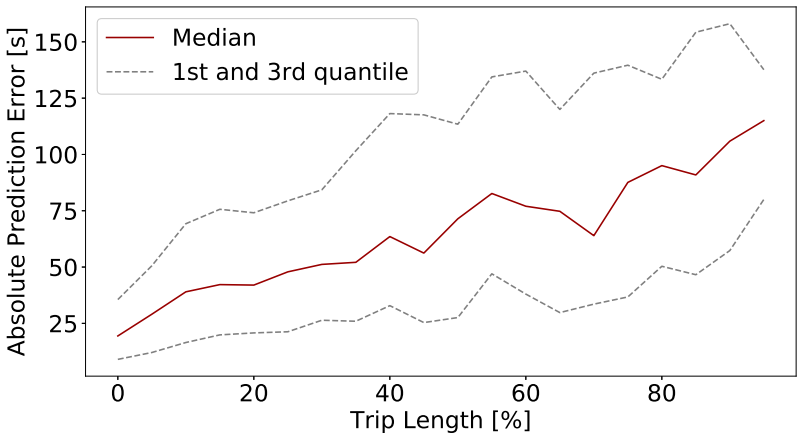

Figure 7: Absolute Error for trips of different lengths show a approximately linear relationship.

Using that a correct prediction is within 60 seconds of the actual arrival time, Figure 8 shows how the accuracy measure changes with different trip lengths. As the Figure illustrates the accuracy decreases for longer trips in a somewhat linear manor and as expected it is easier to correctly predict shorter trips compared to longer.

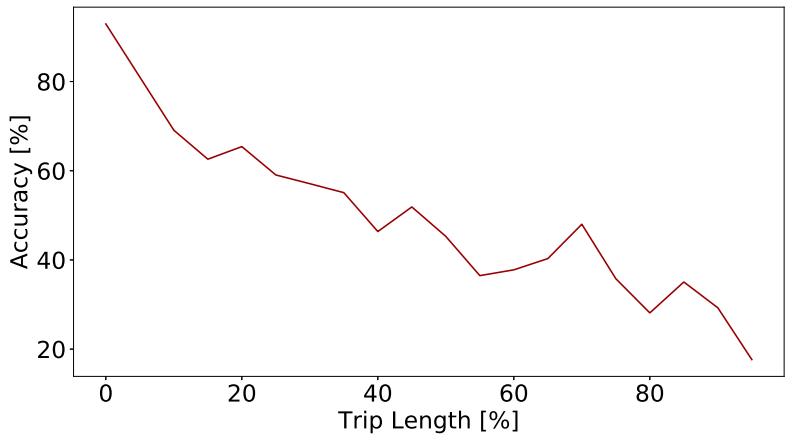

Figure 8: Accuracy measure for trips of different lengths.

In Figure 9 the result of models trained with two loss functions, MAE loss and asymmetric loss are presented. It shows that the asymmetric loss behaves as desired and skews the model towards predicting earlier.

The results when using a model with truncated loss yielded the same results as when using MAE loss, hence was not further investigated.

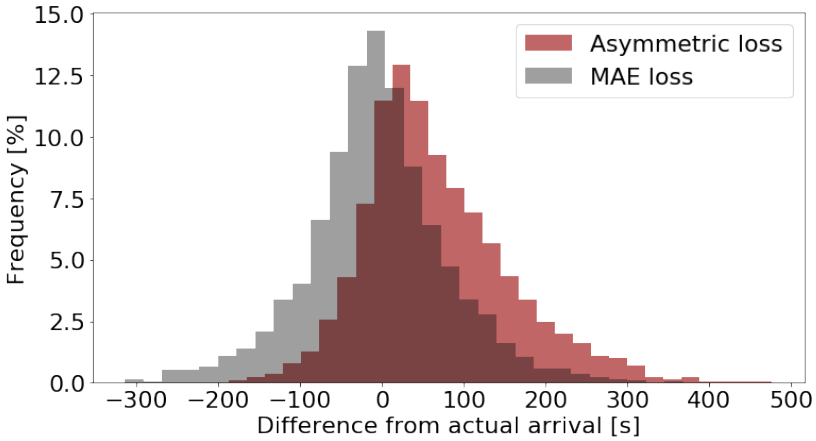

Figure 9: The distribution of prediction difference for two different loss functions.

In Table 2 the result for different hyperparameters in the model are shown. Different network sizes and batch sizes where investigated. The highest accuracy was yielded from a hidden layer size of 100 and a batch size of 250.

Table 2: Neural network accuracy when varying batch size (BS) and hidden layer size (HLS).

HLS \ BS	100	250	500
25	55.6	59.0	57.6
50	56.2	58.4	58.6
100	58.2	59.9	59.8
200	57.4	58.9	59.0

Table 3 presents a comparison of the different symmetric loss functions as well as the effect of batch normalisation on the MAE loss. The MAE with batch normalisation outperforms both alternatives on the validation data. One interesting thing to note is that batch normalisation improved the performance both on the validation and training data sets. This goes against the notion that it would act as regularisation since that would degrade the training performance. One explanation of this is that the smoothing effect of the batch normalisation increases the training performance more than the regularisation degrades it, leading to a net increase in training accuracy.

Table 3: Comparison of different symetric losses as well as the effect of batch normalisation. Values from train and validation data. Results from validation data are presented in parenthesis.

	Accuracy [%]	MAE [s]
MSE with bn	64.6 (59.8)	57.1 (67.4)
MAE without bn	69.9 (58.3)	54.2 (68.4)
MAE with bn	72.8 (60.9)	47.5 (65.4)

5.1 Discussion of Computational Cost

In general the training of a neural network is relatively computationally heavy compared to other machine learning techniques. The actual execution of a network prediction however only consists of a few matrix multiplications and therefore can be done even on modern mobile devices or directly in web browsers. This opens the opportunity to do the actual model executions on the edge devices, leading to reduced server costs for UL. As it currently stands the great computational burden is preprocessing, both of the training data and real time execution of the model. However not much time has been spent on optimising this and since different predictions are independent in terms of input data, the task is very parallelisable. One could even imagine doing the real time preprocessing on the edge devices.

6 Conclusions

We have implemented a neural network model to predict bus arrivals from GPS data. Most of the project time was spent on preprocessing the GPS data collected from Upplands Lokaltrafik. We successfully used batch normalisation to improve the model both in terms of training time and final accuracy. Two custom loss functions, truncated loss and asymmetric loss where evaluated. The truncated loss did not increase performance while the asymmetric loss skewed the model towards earlier predictions as expected.

The model performed slightly better in terms of arrival accuracy than the baseline random forest model and previous neural network models which did not use GPS data (see, [1]). We conclude that real time position data is relevant when predicting bus arrival times. The implemented framework is computationally feasible even for significantly larger systems of bus lines.

6.1 Future Work

To further increase the performance of the model the first step would be to train with much more data. This would likely enable the model to better characterise outliers in the dataset and hopefully increase the relevance of seasonal features like weather. It would also be interesting to further investigate the use of a recurrent neural network. To better capture the traffic situation and thereby increase the accuracy one could imagine using data like city event information. Extending the project to deal with more then one line would open the opportunity to exchange traffic information between routes that drive on the same physical streets. Another interesting point is to investigate the generality of this kind of models. Are they equally applicable on regional buses over larger distances?

References

[1] Johan Rideg, Max Markensten, 2019, 'Are we there yet?: Prediciting bus arrival times with an artificial neural network', (Unpublished Bachelor Thesis), Uppsala Universitet, Uppsala, Sweden.

[2] Christoffer Fors Johansson, 2019, 'Arrival Time Predictions for Buses using Recurrent Neural Networks', (Unpublished Master Thesis), Linköping University, Linköping, Sweden.

[3] Wei Fan, Zegeye Gurmu, 2015, 'Dynamic Travel Time Prediction Models for Buses Using Only GPS Data', International Journal of Transportation Science and Technology, vol. 4, issue 4, pp. 353-366.

[4] Raspberry Pi [Image on internet], edited. Available from: `https://bit.ly/2vbEotG`. Retrived 2020-02-06.

[5] Christopher M. Bishop, 1995, "Neural Network for Pattern Recognition", Aston University, Burmingham, UK.

[6] Boris Hanin, Mark Sellke, 2018, "APPROXIMATING CONTINUOUS FUNCTIONS BY RELU NETS OF MINIMAL WIDTH", *arXiv:1710.11278*.

[7] SXavier Glorot, Antoine Bordes and Yoshua Bengi, 2011, "Deep sparse rectifier neural networks", *AISTATS*.

[8] Diederik P. Kingma, Jimmy Ba, 2014, "Adam: A Method for Stochastic Optimization" *arXiv:1412.6980*.

[9] Sergey Ioffe, Christian Szegedy, 2015, Batch Normalization: "Accelerating Deep Network Training by Reducing Internal Covariate Shift", *arXiv:1502.03167*, Google Inc.

[10] Swedish Meterological and Hydrological Institute weather API. *https://opendata.smhi.se/apidocs/metobs/index.html*. Retrieved December 2019.

[11] UL Bus route API, *https://api.ul.se/api/v3*. Retrieved December 2019.