NITC
NATIONAL INSTITUTE for TRANSPORTATION and COMMUNITIES

Final Report 1299
August 2021

Photo by Elena Muraeva/iStock

Green Waves, Machine Learning, and Predictive Analytics: Making Streets Better for People on Bikes

Stephen Fickas, Ph.D.

UNIVERSITY OF OREGON

Green Waves, Machine Learning, and Predictive Analytics

Making Streets Better for People on Bikes

Final Report

NITC-RR-1299

by

Stephen Fickas
University of Oregon

for

National Institute for Transportation and Communities (NITC)
P.O. Box 751
Portland, OR 97207

August 2021

Technical Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NITC-RR-1299		

4. Title and Subtitle	5. Report Date
Green Waves, Machine Learning, and Predictive Analytics: Making Streets Better for People on Bikes	August 2021
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Stephen Fickas: https://orcid.org/0000-0001-7816-0731	

9. Performing Organization Name and Address	10. Work Unit No. (TRAIS)
University of Oregon, Eugene OR	11. Contract or Grant No. NITC 1299

12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered
National Institute for Transportation and Communities (NITC) P.O. Box 751 Portland, OR 97207	14. Sponsoring Agency Code

15. Supplementary Notes

A Jupyter notebook is provided as a companion piece to support both replication and further experiments (Widder et al., 2019). See Project's Jupyter notebook.

16. Abstract

This project focuses on giving bicyclists a safer and more efficient path through a city's signalized intersections. It builds on a prior NITC project that tested an app for a fixed-time corridor. The goal of this project is to lay the groundwork for extending this earlier app to include actuated signals. Two machine-learning algorithms are introduced that have a good track record with time-series forecasting: LSTM and 1D CNN. The algorithms are tested on data captured from a busy bike corridor on the south end of the University of Oregon campus. A specific actuated intersection is identified on this corridor and real-time data is collected from it. The algorithms are trained on the data and evaluated. The results show that both algorithms can reach 85% accuracy and can predict on a single sample within roughly one second. While these results are encouraging in terms of adding a prediction component to the existing app, a closer look at Precision and Recall is more mixed. A means of computing a Precision-Recall tradeoff is discussed.

17. Key Words	18. Distribution Statement
Bicyclists, actuated signals, time-series forecasting, machine learning, precision-recall tradeoff, green wave phone app	No restrictions. Copies available from NITC: www.nitc-utc.net

19. Security Classification (of this report)	20. Security Classification (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	23	

ACKNOWLEDGEMENTS

This project was funded by the National Institute for Transportation and Communities (NITC; grant number 1299), a U.S. DOT University Transportation Center. We are grateful to the City of Eugene Transportation Office, and Andrew Kading in particular. Two graduate students laid the groundwork for this project: a big thanks to Brian Williams and Ben Bennett. We also thank Professor Schlossberg for his valuable insight.

DISCLAIMER

RECOMMENDED CITATION

Fickas, Stephen. *Green Waves, Machine Learning, and Predictive Analytics: Making Streets Better for People on Bikes*, NITC-1299. Portland, OR: Transportation Research and Education Center (TREC), 2021.

TABLE OF CONTENTS

APPENDICES

None.

LIST OF TABLES

NONE

LIST OF FIGURES

EXECUTIVE SUMMARY

This project is a follow-on to two prior NITC projects: (1) *V2X: Adding Bikes to the Mix*, NITC-ED-1027 (referenced hereafter as V2X) and (2) *FastTrack: Allowing Bikes To Participate In A Smart-Transportation System*, NITC-1160 (referenced hereafter as FastTrack). The overall goal of these two prior projects and the current project is to give bicyclists a safer and more efficient use of a city's signaled intersections. The V2X project focused on giving bicyclists a virtual call button that functioned on a phone app. From this project, we were able to collect detailed real-time data on an actuated signal on a busy bike corridor near the University of Oregon campus. The FastTrack project focused on giving bicyclists a GLOSA (Green Light Optimized Speed Advisory) or more colloquially, a green wave. The project focused on non-actuated (i.e., fixed-time) signals along a second busy bike corridor near the University of Oregon campus. The current project uses ideas from both prior studies: (1) It uses the data collected from the actuated signal to train and test two machine-learning algorithms. (2) It sets the groundwork to extend the FastTrack app to include both non-actuated and actuated signals, a situation that bicyclists are likely to encounter. In particular, this report summarizes our attempts to use machine-learning algorithms to predict the next phase of an actuated signal given a look-back of K previous phases, where K is a parameter that can be explored. The long-term goal is to give a bicyclist real-time information on whether to slow down, speed up, or maintain speed in order to make a green. Our earlier study did this for non-actuated signals. We are now interested in extending that app to actuated signals as well. This study is the first step toward that goal.

1.0 INTRODUCTION

1.1 BACKGROUND

The project builds on a prior app that was designed for Green Light Optimized Speed Advisory (GLOSA). This is more colloquially known as keeping a vehicle in the green wave: you are at a location and moving at a speed that will allow you to (theoretically) have a green light at each intersection you encounter along a corridor. If you are not in the green wave, then advice will be given on adjusting your speed (Suzuki & Marumo, 2020). GLOSA-capable systems are starting to appear in cities across the U.S. (e.g., Dallas; Denver; Gainesville, FL.; Houston; Kansas City, KS; Las Vegas; Los Angeles; New York City; Orlando, FL.; Phoenix; Portland, OR; San Francisco; and Washington, D.C.). Figure 1.1 shows a simple design of a GLOSA driver interface built into the car's speedometer. As part of the FastTrack project, our interest was in providing an analog of Figure 1.1 for bike riders. We assume that a bike rider has a phone mounted on the handlebar and that our app is active and visible. We expect the app to provide speed

adjustments (increase speed, decrease speed, hold steady) to allow the rider to pass through the upcoming signal safely and without stopping.

Figure 1.1: A GLOSA interface for motor vehicles. The green band moves to place the driver in a green wave.

1.2 FASTTRACK TEST SITE: 13TH AVENUE CORRIDOR

Our FastTrack project focused on a busy bike corridor that leads into the west campus entrance of the University of Oregon. The corridor lies along W 13th Avenue from Willamette Street (on the west end of the corridor) to Hilyard Street (on the east end of the corridor). The corridor is roughly .5 miles long and one-way east for both cars and bikes. It has six fixed-time, semi-coordinated signals (discussed in more detail below). The speed limit along the corridor is 25 mph. Figure 1.2 shows a satellite view of the corridor with red dots denoting signals. Figure 1.3 shows a street-level view looking east on the corridor and approaching Willamette (1) with the bike lane on the right. The stop-line in Figure 1.3 is the start of our trials.

Figure 1.2: The satellite view of the entire corridor. The start is signal 1 and the end is signal 6.

Figure 1.3: The view approaching the first signal on the corridor. The bike lane is on the right.

The corridor is semi-coordinated in that coordination exists between two pairs of signals but not for the entire corridor. In particular, the pair of signals Willamette (1) and Oak (2) are coordinated. The pair of signals High (3) and Pearl (4) are also coordinated for the speed limit. These gaps are **not** coordinated: between Oak (2) and High (3), between Pearl (4) and Patterson (5), and between Patterson (5) and Hilyard (6). Given that all signals are fixed-time, in theory they will all come into and out of alignment during the day.

We view the semi-coordinated property of the corridor as a feature rather than a bug. It places more value on a GLOSA app, which needs to do real-time adjustments based on the shifting alignment in the gaps. In particular, if the entire corridor was coordinated, then a speed of 25 mph would typically allow a motorist to stay in the green wave for the entire corridor. But that is not the case with our test corridor. It is also not the case that we can reasonably expect a bike rider to maintain a 25 mph pace even if it maintained the green wave; a more reasonable biking speed is roughly half that. In summary, it becomes important to give a bike rider support in this rather challenging corridor.

1.3 FASTTRACK GOALS

Our primary goal was to give bike riders along the 13th Avenue bike corridor a real-time display that shows GLOSA information. In particular, we want to let the bicyclist know, given their current location, direction and speed, whether they will reach the next signal in their path with a green light (i.e., they are in the green wave for that signal). If they will not get a green given their current speed, we want to give them further information on *reasonable* speed adjustments they can take to bring them back into the green wave. Adjustment advice will specify whether to increase or decrease their speed and by how much.

1.4 AN EFFECTIVE USER INTERFACE

We would like the interface to be effective (it provides speed adjustments that give the rider the best chance of making green lights); reasonable (it does not ask the bike rider for super-human performance); and safe (it does not force the rider to attend to the interface in a way that distracts from situational awareness). We chose to use two modes of information delivery based on our findings from a previous three-year NSF study of user interfaces for transportation information. Our earlier study found a visual interface slightly easier to use, but also distracting in a way that audio is not. Some users could use the visual interface effectively without being distracted while others worked most effectively with audio only (Fickas et al., 2008;Robinson & Fickas, 2009; Fickas et al., 2013).

We chose to use a cell phone as the interface device. For testing, we placed the phone in a holder on the handlebar and provided both visual and audio information to the bicyclist (see Figure 1.4). However, we also ran several extra tests with the phone in a backpack to test the audibility of the audio interface alone.

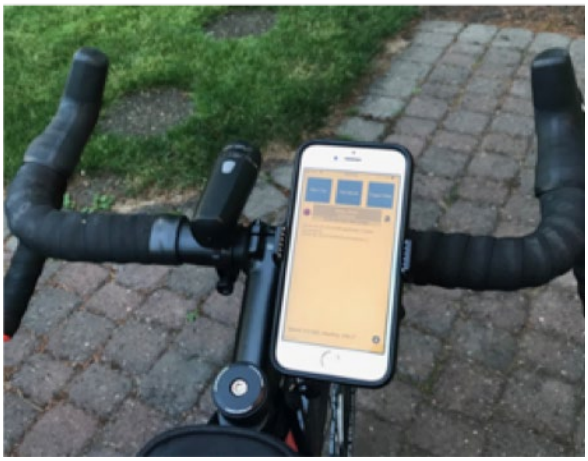

Figure 1.4: The project supplied the phone and holder for the test trials.

The visual interface we used in our trials is shown in Figure 1.5. It consists of two separate information displays. The first is a large area for a set of icons we developed for the project. The figure shows the checkmark icon. All possible icons are shown in Figure 7. When the user is stopped at a signal the X icon appears and remains until the light turns green (as predicted by the app).

Prior to the field trials, we tested in a virtual environment (Masud & Fickas, 2011) to narrow interface options. For the trials we linked both small-adjustment arrows to changes of 2 mph or less. The large arrows we linked to changes greater than 2 mph . The one exception is when a rider is starting from a full stop. In this case the app will display the small up-arrow icon for five seconds before calculating the actual adjustment. All of these values can be changed to a user's preference but were held steady in our trials.

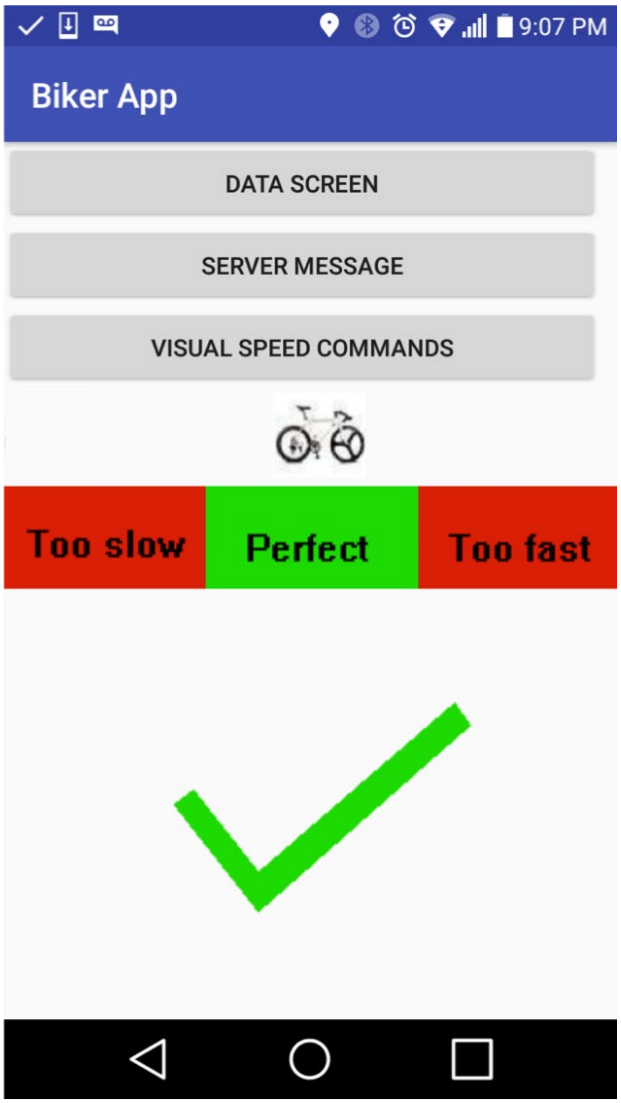

Figure 1.5: Delivers GLOSA information to a bike rider on a corridor. In the case shown, the rider is in great position to catch the next green.

The second visual display is of a bicycle that travels back and forth across the top of the three text boxes. It is meant to give a more fine-grained picture of where a bike rider is in terms of the green-wave interval. For instance, if the bike is straddling the “Too slow” and “Perfect” text boxes, the user can see that they are on the edge of falling out of the green-wave interval. When the user is stopped at a red light, the bicycle parks on the left-hand edge of the display.

The audio interface consists of six alternative messages: (1) “increase speed” (2) “reduce speed” (3) “in green wave” (4) “impossible” (5) “entering corridor” and (6) “exiting corridor.” After initial tests, the repetitiveness of the adjustment messages was set at every five seconds. The green-wave message was delivered immediately when transitioning into the green wave and then every 10 seconds. The impossible message was delivered just once, as were the entering and exiting messages. We also note that these are all settable to each individual rider’s preference. In particular, at least one

early tester would have liked to be reminded they were doing well more often; hearing the green-wave message made riding “more fun.” However, we eventually set them to the values above for our final trials.

The tabs on the top of the screen are for use in debugging and were not used in the trials.

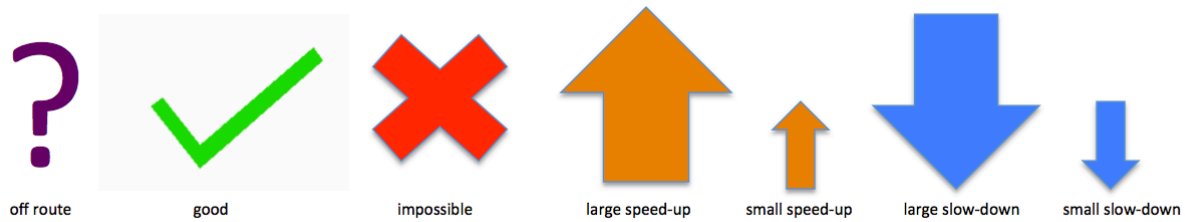

Figure 1.6: The possible icons available to the app to convey GLOSA adjustments to the rider.

2.0 CURRENT PROJECT GOALS

Our long-term goal is to extend the FastTrack app described in the Background section to include actuated signals along a corridor. This project takes a first step by evaluating the effectiveness of machine-learning algorithms to predict the next phase of an actuated signal on a busy bike corridor, given information about the past K phases. In essence, this is what is called a time-series forecasting problem. If we find forecasting success here, then we can begin to incorporate these algorithms into a more comprehensive GLOSA app (post-grant).

The project used data captured during the prior V2X project. It consists of phase-change data for a complicated intersection that plays a key role in a bike corridor. The intersection has eight separate phases, all callable, that serve vehicles, bicyclists, pedestrians, and buses, all in various combinations. The data was taken from the month of June 2018. June is typically a heavy biking month near campus and, hence, a good test month for us.

In the following sections we will describe the field test site, the two machine-learning algorithms explored, the results, and conclusions.

2.1 FIELD TESTING LOCATION

Our data comes from the intersection of Alder and 18th in Eugene, which is part of a busy bike corridor leading to and from the south end of the UO campus. Loop detectors and advanced loop detectors currently exist in both directions on Alder to recognize the presence of bicycles and vehicles; there are also pedestrian call-buttons on all four corners. Figures 2.1 and 2.2 give different views of the intersection.

Figure 2.1: Intersection of Alder and 18th (looking south on Alder) with signal phases for bike only (left) and car only (right) and the terminal loop detector bottom left (many people on bikes wait in the crosswalk not knowing what the bike loop detector symbol is for).

Figure 2.2: Intersection of Alder and 18th (looking north on Alder).

For our study intersection, there are eight phases possible. Three of these are what we call “bike friendly.” Through a combination of pedestrian and bike greens, these three phases allow a bike rider to travel through the intersection without stopping.

2.2 THE DATA

The City of Eugene gave us access to the June data (12 days) in real time from the McCain Transparity server monitoring the intersection (Transparity TMS). We captured and then uploaded the data to a Python pandas DataFrame as seen in Figure 2.3, which shows a sample of five rows taken from the table. As shown, a new row was generated on each phase change. This could include a complete change to a new phase or simply an extension to the current phase. Note that a value of 0,0 for x and y designates a change to yellow. This will be wrangled out and not counted as an actual phase.

	date	time	x	y
	06/15/2018	2:04:43	0.0	0.0
	06/15/2018	2:04:47	4.0	0.0
	06/15/2018	2:04:52	0.0	0.0
	06/15/2018	2:04:56	2.0	6.0
	06/15/2018	2:05:05	0.0	0.0

Figure 2.3: Data as captured from McCain Transparity server.

The raw table has 42,920 rows (i.e., separate phase changes), yellows included.

2.3 UNIVARIATE VERSUS MULTIVARIATE

In time-series forecasting, two types of data are possible. The simplest is univariate data. Think of predicting the temperature based on the past five days' temperatures. So, a single variable that acts as both an independent and dependent variable. But you can easily view the same temperature-prediction problem from a more comprehensive approach by considering more variables (e.g., the humidity, barometric pressure, cloud cover) over the last five days. Using two or more variables to predict temperature turns it into a multivariate forecasting problem.

In our case, we have two or more variables we could use, including date and time of day. With some wrangling, we could also add a new column for the day of week. We potentially could branch out and merge in weather data to a new column. In short, we could use multivariate forecasting approaches. In general, these will outperform univariate approaches given there is more information available to make predictions. However, they also rely on having continuous access to multiple information sources. While our current data has no gaps, we have worked with controllers in the past where things like time-stamps are randomly dropped or mangled. Hence, we decided to first test our algorithms on a simple univariate problem because it relies on fewer sources of (potentially unreliable) information.

2.4 WRANGLING TO UNIVARIATE

First, we removed rows of phase 0,0 that represent a phase change to yellow. Next, we added a new column that encoded each separate phase as a binary number: 1 for bike friendly and 0 for not. The three phases identified as bike friendly are 2,6 and 2,0 and 6,0. These three-phase combinations give a bike rider a green for the intersection. The remaining five combinations force the rider to stop and wait. Finally, we dropped all other columns. We now have a univariate dataset. Figure 2.4 shows the first five rows of the resulting table (i.e., an interleaving of bike friendly and not). We ended up with a sequence of 22,535 rows in the table after removing yellows.

	phase
0	0.0
1	1.0
2	0.0
3	1.0
4	0.0

Figure 2.4: Data wrangled to univariate form.

2.5 CHOICE OF LOOK-BACK

We next transformed the pandas table into a dataset that is suitable for TensorFlow. At this point we had to choose the look-back value: how many phases should we look-back to predict the next phase? This value can range from one (look-back to just the previous phase) to the hundreds. After initial exploration, we choose seven: use the past seven phases to predict the next.

3.0 METHODOLOGY

We chose to explore two separate machine-learning algorithms. Both have a good track record with time-series forecasting: One-Dimensional Convolutional Neural Nets (1D CNN for short) and Long Short-Term Memory models (LSTM for short). We chose to use the Python TensorFlow library (TensorFlow Overview), which has good support for both algorithms. We recommend two tutorials from Brownlee's *Deep Learning and Time Series* for those wishing to dive a little deeper into the two algorithms: 1D Convolutional Neural Network Models for Human Activity Recognition and How to Develop LSTM Models for Time Series Forecasting.

3.1 THE THREE METRICS USED

Our experience is that bike riders become more annoyed at being mistakenly told they will get a green (and then potentially required to slam on their brakes because of an unexpected red) than mistakenly told they cannot make a green (and then potentially missing a green). Hence, we put more value on correct bike friendly predictions of 1. We chose to use three metrics because of this: Precision, Recall and Accuracy. In words, Precision is concerned with "when the model does predict 1, how often is it correct?" As a complement, Recall asks "for all the actual 1s, how many did the model get correct?" A high Precision score (with 1.0 being tops) says that the model is not prone to have the rider slamming on brakes. A high Recall score (with 1.0 being tops) says that the rider is not missing many greens. Finally, Accuracy is simply the number of correct predictions.

3.2 LSTM

The general idea behind an LSTM model is somewhat the opposite of a CNN model (discussed shortly). Whereas a CNN model uses filters to actively search for what to glean from the data, an LSTM stays put and lets the data come to it through a sequence of cells. Figure 3.1 shows a diagram from an LSTM layer with three cells. The cell itself has mechanisms for both remembering and forgetting what it has seen in the past.

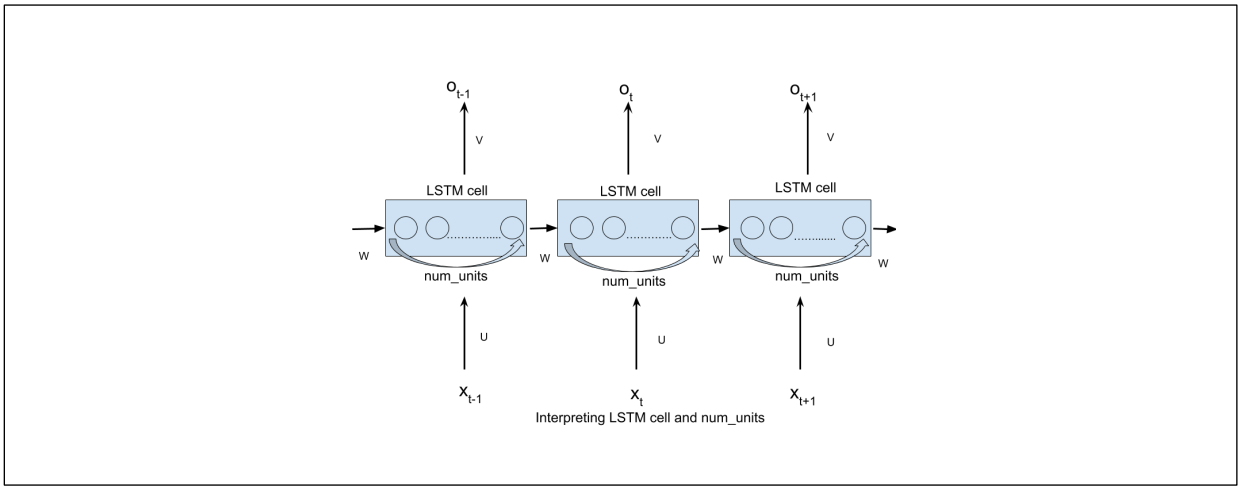

Figure 3.1: Cells of an LSTM.

3.3 CHOICE OF LSTM ARCHITECTURE

One of the complicating factors of using neural net algorithms is their large hyperparameter space: many choices are left to the user; there are no hard and fast rules on what choices work best. What you are left with is a large exploration space that is typically costly to explore. After this exploration process, we chose the model architecture shown in Figure 3.2.

```
lstm_size = 50
l_model = Sequential()
l_model.add(LSTM(lstm_size, activation='selu',
              input_shape=(n_timesteps,1), return_sequences=True))
l_model.add(Dropout(0.2))
l_model.add(LSTM(lstm_size, activation='selu'))
l_model.add(Dropout(0.2))
l_model.add(Dense(20, activation='selu'))
l_model.add(Dropout(0.2))
l_model.add(Dense(10, activation='selu'))
l_model.add(Dropout(0.2))
l_model.add(Dense(1, activation='sigmoid'))
l_model.compile(optimizer='adam',
            loss='binary_crossentropy',
            metrics=['accuracy'])
```

Figure 3.2: LSTM architecture.

As an explanation:

- We are using two LSTM layers of 50 units/cells each. Both use the selu activation function (Scaled Rectified Linear Unit – see Why Scaled?). The return_sequences=True code is needed when stacking like this.
- The dense portion has two hidden layers, one with 20 nodes followed by one with 10 nodes.
- The output layer has one node using sigmoid.
- There are Dropout nodes interleaved throughout the layers, each with a percentage of .2.
- We compile the whole model using standard parameters.

3.4 LSTM TRAINING

We used the same 80/20 split for training and testing data on both 1D CNN and LTSM models. This gave us 18,021 training rows and 4,500 testing rows. We also chose 5 epochs with a batch size of 1. Note the training time of the LTSM as configured was eight minutes or **96 seconds per epoch**.

```
training = l_model.fit(train_x, train_y, epochs=5, batch_size=1)
CPU times: user 8min
```

Figure 3.3: LSTM training.

3.5 LSTM RESULTS

```
Precision:  0.7545126353790613
Recall:     0.9776507276507277
Accuracy:   0.8544444444444445
1s 3ms/step
```

Figure 3.4: LSTM evaluation.

The Recall score is good. We would expect a bike rider to miss just a few greens. The Precision score says we correctly predict 1 roughly 76% of the time and incorrectly predict 1 24% of the time. So, 24% of the time we could expect the need for braking.

3.6 1D CNN

The general idea behind a 1D CNN is that of moving filters or convolvers. Multiple filters move over a time series looking for patterns. Given multiple filters, over training each tends to specialize in finding specific features in the data. When combined, they often prove highly effective in prediction, especially over non-linear data. Figure 3.5 illustrates a simple 1D CNN where the filters are called “feature maps.” As shown, the back end to a convolution layer is a normal neural net.

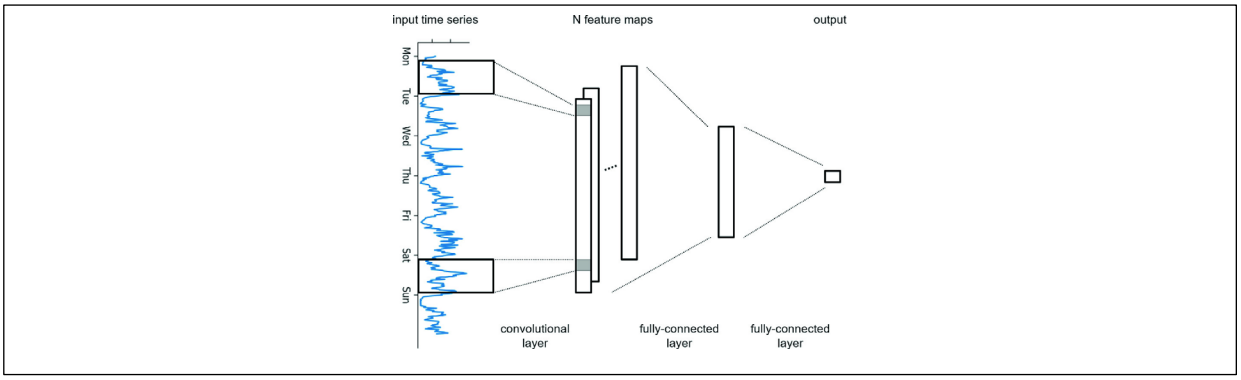

Figure 3.5: 1D CNN.

3.7 CHOICE OF 1D CNN ARCHITECTURE

After exploration, we chose the model architecture shown in Figure 3.6.

```
c_model = Sequential()
c_model.add(Conv1D(filters=16, kernel_size=4, activation='selu',
                   input_shape=(n_timesteps,1)))
c_model.add(Dropout(0.4))
c_model.add(Conv1D(filters=8, kernel_size=3, activation='selu'))
c_model.add(Dropout(0.4))
c_model.add(Conv1D(filters=4, kernel_size=2, activation='selu'))
c_model.add(Dropout(0.4))
c_model.add(Flatten())
c_model.add(Dense(20, activation='selu'))
c_model.add(Dropout(0.2))
c_model.add(Dense(10, activation='selu'))
c_model.add(Dropout(0.2))
c_model.add(Dense(1, activation='sigmoid'))
c_model.compile(optimizer='adam',
            loss='binary_crossentropy',
            metrics=['accuracy'])
```

Figure 3.6: 1D CNN architecture.

As an explanation:

- We are using three separate convolution layers: the first has 16 filters of size 3, (i.e., it looks at three separate phases (in series) at once, then slides to the next). It has a default stride of 1: it moves down one row at a time. Given seven rows in the look-back series, it will examine five combinations of 3 in series. All use the selu activation function (Scaled Rectified Linear Unit).
- The flatten layer prepares the data for a normal (dense) neural net.
- The dense portion has two hidden layers, one with 20 nodes followed by one with 10 nodes.
- Similar to the LTSM, the output layer has one node.

3.8 1D CNN TRAINING

We used the same 80/20 split as with the LTSM. Note that the training time was roughly 150 seconds with this configuration or **30 seconds per epoch** (three times faster than with the LSTM).

```
training = c_model.fit(train_x, train_y, epochs=5, batch_size=1)
CPU times: user 2min 28s
```

Figure 3.7: 1D CNN training.

3.9 1D CNN RESULTS

The results are nearly identical to the LSTM scores. Our takeaway is that we may have maxed out what a deep-learning model can do with this dataset. However, there is still room to explore, which we will discuss in the next section.

```
Precision: 0.7461180124223602
Recall: 0.998960498960499
Accuracy: 0.8542222222222222

0s 1ms/step
```

Figure 3.8: 1D CNN evaluation.

4.0 PRECISION-RECALL TRADEOFF

We decided to do further exploration on Precision and Recall. From our experience, a Precision score of .75 is lower than we would like: it would require a bike rider to unexpectedly have to stop, potentially quickly, 25% of the time. The one thing we can tradeoff is Recall (i.e., the ability to give bike riders the opportunity to make the maximum number of greens). The general idea is to introduce a threshold that can be varied on the raw sigmoid values from a model. As a reminder, we are using sigmoid to give us a value from 0 to 1, e.g., .2, .45, .9. We eventually want to transform the raw sigmoid value to a binary 0 or 1 and, hence, match up with actual label value of 0 or 1. We can introduce a decision rule for this transformation that includes a threshold. Figure 4.1 shows the general form of this type of rule in Python, where raw_sigmoid is the list of predictions obtained from a model as values between 0 and 1.

Figure 4.1: A decision rule with a threshold.

```
binary = [1 if r>=threshold else 0 for r in raw_sigmoid]
```

By default, the threshold has a value of .5: if the sigmoid output is greater than or equal to this value, a 1 is predicted otherwise a 0 is predicted. But we can increase the threshold value to increase Precision (and typically decrease Recall). We explored this tradeoff with the output of the LTSM model. Our results are shown in Figure 4.2.

Threshold	Precision	Recall	Accuracy
0.50	0.746118	0.998960	0.854222
0.55	0.750884	0.993243	0.856222
0.60	0.764235	0.941788	0.850889
0.65	0.772788	0.903326	0.845111
0.70	0.780344	0.825364	0.826000
0.71	0.780344	0.825364	0.826000
0.72	0.780344	0.825364	0.826000
0.73	0.781586	0.794179	0.817111
0.74	0.795455	0.764033	0.815111
0.75	0.795455	0.764033	0.815111
0.76	0.810135	0.656445	0.787333
0.77	0.810135	0.656445	0.787333
0.78	0.822047	0.542620	0.754222
0.79	0.918033	0.029106	0.583778
0.80	0.933333	0.029106	0.584000
0.85	1.000000	0.016112	0.579333

Figure 4.2: Threshold exploration in table form.

Or in graph form in Figure 4.3.

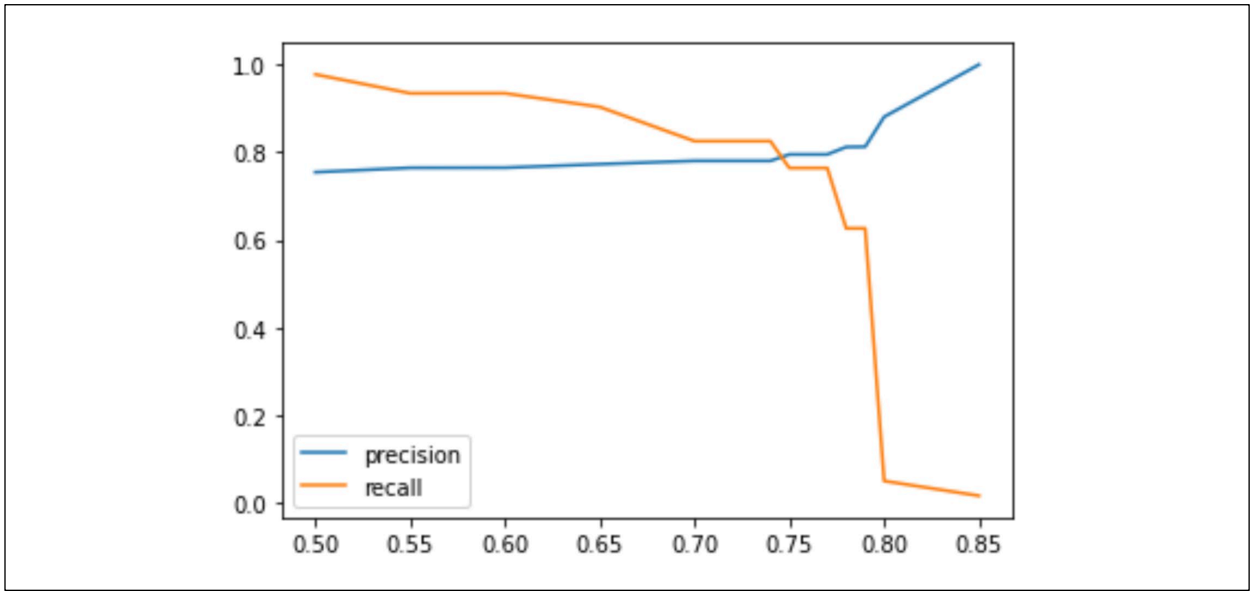

Figure 4.3: Threshold exploration in graph form.

At this point we are making subjective choices. If we highly prize Precision, then perhaps we should go with a threshold of .85 with a Precision value of 1.0 (i.e., we are always correct when we predict the rider will make it). Of course, the accompanying Recall is below .02 (i.e., we are catching less than 2% of all greens). What is interesting about introducing a decision rule with a threshold is that the threshold can be tailored to individual riders' tastes. Perhaps Smith is ok with less Precision if it provides better Recall. In that case, we could set the FastTrack app to include a threshold of .5 for

Smith. Jones, on the other hand, is worried about sudden stops with their poor brakes. We can set the threshold for Jones to .79 or above.

5.0 CONCLUSION

Our results are mixed for the 12 days in June 2018 that we were given access to. We were able to predict the next phase with two separate time-series forecasting algorithms with roughly 85% accuracy given a look-back of 7. And both algorithms were able to predict a single sample within one second, which is reasonable for inclusion in the FastTrack app. However, looking more closely at Precision and Recall, our values for Precision, at roughly 75%, were a bit disappointing. We did explore a means of increasing Precision at the cost of Recall by introducing a threshold that we could vary. And argued that this can be used to tailor the FastTrack app to different users' tastes.

We believe we are in the ballpark of being acceptable in terms of adding a prediction component to our existing FastTrack app. This would open up green-wave capability for non-fixed-time intersections. Our plans for next steps are:

1. Gain access to a dataset with a larger range of days, perhaps an entire season. It appears we will be able to do this for the Naito Parkway corridor in Portland. This corridor contains multiple actuated intersections to draw data from. Typically, more data leads to stronger results when looking at machine-learning algorithms.

2. We believe an effective step will be to move to a multivariate dataset that includes date and time, and perhaps weather as well. This would not be a huge change to data preparation. And it may allow a single model that covers all four seasons. The downside is that of mangled or corrupted data from these new data sources. However, looking at our upcoming access to Naito Parkway, all new (modern) controllers are being installed along the corridor, suggesting a clean real-time feed.

Finally, our app requires a real-time feed from upcoming signals on the bicyclist's path. Cities with older equipment or with older Traffic Management Systems (TMS) may not be able to provide this feed. We can relate to this given our challenges getting this feed from the Eugene TMS. However, we are optimistic. We expect that as cities replace older equipment and bring on a modern TMS, they will be fully capable of using a FastTrack app that is effective with both fixed and actuated intersections, giving their biking community green-wave opportunities.

Note that following (Widder, et al., 2019), we have made our code available in a Colab Jupyter notebook for those interested in replicating our work or exploring further: Colab notebook.

6.0 REFERENCES

Fickas, S., Sohlberg, M., Hung, P., (2008) Route-following assistance for travelers with cognitive impairments: A comparison of four prompt modes, *Int. J. Human–Computer Studies*, Volume 66, Issue 12, December 2008, Pages 876-888

Fickas, S., Lemoncello, R., Sohlberg, M. (2013) Requirements Engineering in a Mobile Setting, In *User Modeling and Adaptation for Daily Routines*, Martín, Haya, Carro (Eds.), Springer

Fickas, S. (2018) *V2X: Adding Bikes to the Mix*. NITC-ED-1027. Portland, OR: Transportation Research and Education Center (TREC), 2018.

Fickas, S., Schlossberg, M. (2019) *FastTrack: Allowing Bikes To Participate In A Smart-Transportation System*, NITC-1160. Portland, OR: Transportation Research and Education Center (TREC), 2019.

Masud, R, Fickas, S. (2011) Virtual Environments for Testing Location-Based Applications, IUI Workshop on *Location Awareness for Mixed and Dual Reality LAMDa'11*, Palo Alto, California, USA.

Robinson, W.N., and Fickas, S. (2009) Talking Designs: A Case of Feedback for Design Evolution in Assistive Technology, in: *Design Requirements Engineering: A Ten-Year Perspective*, K. Lyytinen, P. Loucopoulos, J. Mylopoulos and W. Robinson (eds.), Springer-Verlag, 2009, pp. 215-237.

So, G. (2019) Should We Abandon LSTM for CNN?, *AI/ML at Symantec*, url.

Suzuki, H., Marumo, Y. (2020) Safety Evaluation of Green Light Optimal Speed Advisory (GLOSA) System in Real-World Signalized Intersection, *J. Robot. Mechatron*., Vol.32, No.3, pp. 598-604, 2020.

Widder, D.G., Sunshine, J., & Fickas, S. (2019) Barriers to Reproducible Scientific Programming, Published in: *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*