Developing Deep Learning Networks for Dynamic Traffic Light Control

Nhan Ngo and Simon Socolow

Bangor High School, Bangor, ME, 04401

Mentors: Barbara Stewart, Theodore Taylor, John Cangelosi, Cary James

I. Abstract

The average Los Angeles commuter spends upwards of 20 work days per year in traffic (9). Though the rise of electric vehicles has reduced greenhouse gas emissions, “a green traffic jam is still a traffic jam” (15). Congestion is caused by a number of theories such as overcrowding, spontaneous generation, interaction with pedestrians, and road work (1). However, another large factor is inadequate traffic light management. Adaptive traffic light control in which light timings are biased based on the number of cars at each node at an intersection have been tested and shown to improve commute speeds by 10% in a 9x16 block section of midtown NYC. Many of these centralized systems use traditional algorithmic control. Recent studies using deep learning improved simulated wait times up to 25%. In this research, a real intersection was chosen to model in Python with observed traffic data. This simulation models vehicles on a grid as an environment to evolve neural network agents that control traffic light timings, whose goal is to minimize wait times. Neuroevolution was used to produce a neural network which reduced wait times 17% compared to the best fixed-light timing that could be made. The implementation of this model has the possibility to reduce costs as it is open source and can run on an SBC, which can accept inputs from detectors such as cameras and can control the lights itself. Combining these controllers across a city can allow the software to find solutions to minimize waiting and increase efficiency.

II. Acknowledgments

We would like to thank Barbara Stewart, Theodore Taylor, and John Cangelosi for their ongoing support through the Bangor High School STEM program. In addition, we would like to thank the traffic engineers at the Bangor Public Works for providing us with insight into the current infrastructure and other information. Finally, we would like to acknowledge Mr. Cary James, former STEM director and mentor.

III. Abbreviations and Definitions

SBC - Single Board Computer
Deep Neural Network (DNN) - any artificial neural network with more than 1 hidden layer
Inbound/toward-intersection lane - a lane that has vehicles feeding into the intersection
Outbound/away-lane - a lane that vehicles go into after navigating the intersection

Table of Contents

1. Introduction and Current Systems

Traffic is a major problem plaguing the modern world. Cars, buses, trucks, trams, etc. compete for space on already cramped roadways. Atmospheric pollutants such as oxides of nitrogen (NO_x) are primarily derived from vehicles, and are exacerbated by increased idle times (6). In 2015 there were approximately 1.3 billion vehicles on the world's roads. That number is expected to increase to over 2 billion by 2040 (27). Several major sources of congestion are laid out as follows:

1. Excess number of vehicles carrying only one or two people.
2. Human factors and spontaneous generation (1)
3. Lack of public transport options or unviability/efficiency for ones that currently exist.
4. Road infrastructure, maintenance, design (number of lanes, turning locations, etc.).
5. Traffic light inefficiencies.

This paper will focus on the fifth point listed, traffic light inefficiencies. Traffic engineers use what are called **phases** and **stages** to determine what lights should be activated in what order (19). A phase is any direction that a vehicle or pedestrian could move, and a stage is a selection of what phases should be activated and for how long. Currently, many traffic signals in less populated areas still use simple clockwork mechanisms. Clockwork mechanism times for each stage are either evenly distributed across all intersecting streets, or time is allocated based on historical data of the number of cars running through the intersection. These methods do not detect the current state of the intersection, and therefore cannot make smart decisions concerning the current state of a network.

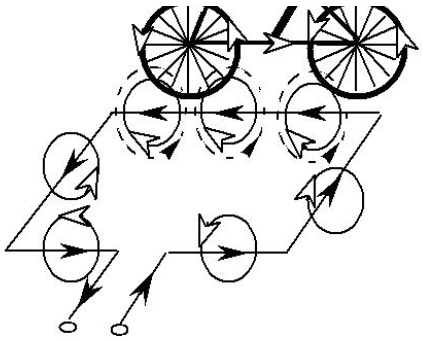


Figure 1.1: *Diagram of induction loop and bike (8)*

Since the 1960's many intersections in the United States have had induction loops installed underground at the stop point of each roadway (8). As shown in Figure 1.1, a metal object traveling over the loop induces a current in the wire below. This current sends a signal to the traffic management system, activating a predesignated protocol to allow the intersecting road with a detected vehicle priority. There have been other iterations on this

method, including using microwave detectors and single-lane cameras. The problem with these systems is that they can only make an assumption about the number of cars at each point in the intersection. They cannot sense whether or not there are 10 cars at one point and 1 at another. This is only a temporary fix to an ever increasing problem.

Many methods have been proposed and tested to replace timed and induction loop influenced intersections. Recently, major strides have occurred in the fields of computer vision and machine learning (2,3,4,5). Modern intersections now incorporate computer vision for detection. However, software for determining optimal timing (adaptive control) is still being researched and improved on. Because of this, researchers have been developing systems for controlled intersections that improve wait times. These methods will be discussed further in order to show the current state of technology and the proposed improvements of this research.

1.2. Functions of a Traffic Controller

The brains of a traffic signal is the controller (21). Traffic controllers are set up inside traffic management cabinets at each intersection (on its own stand or on an electrical pole), which outputs the power that lights up the corresponding lights. A traffic engineer may set the phases that are to run at that intersection in the memory of the controller, and the controller will choose which phase based on a timing sequence or influenced by detection devices such as induction loops and cameras discussed previously. The phase selection and duration can be catered to the time of day or a particular day in the year where traffic loads may be different than normal. Traffic management boxes also have redundancies hard-wired into boards, which can set the lights to default to red in the case of a system failure as to prevent a possible crash (21).

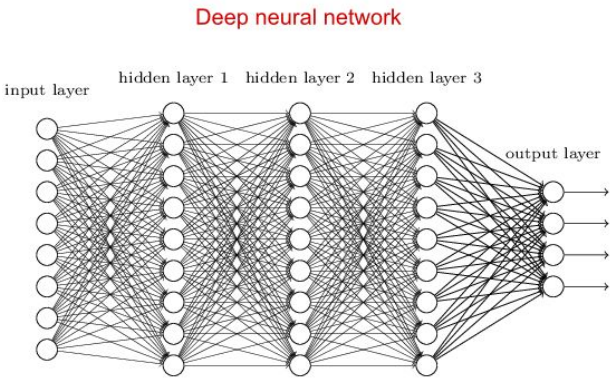
The functions of a traffic controller are outlined in the Traffic Control Systems Handbook by the U.S. Department of Transportation, and are shown below (21).

- Can Control:
    - single intersection
    - closely spaced multiple intersections
    - midblock crosswalk
- Electrically switches signal indications:
    - red
    - yellow
    - green
    - WALK
    - DON'T WALK
    - other
- Assures appropriate right-of-way assignments in accordance with pretimed or actuated intervals or phases
- Times fixed clearance intervals such as:
    - flashing DON'T WALK
    - yellow
    - red clearance
- Times greens and green arrows for:
    - fixed-duration (pretimed control)
    - variable duration (up to a predetermined maximum) according to traffic demand (actuated control)
- Times special function timed intervals such as:
    - lane controls
    - turn controls
    - blank out signs

This study focuses primarily on a single intersection situation using actuated control. In this study, the purpose of a traffic controller has not changed itself (i.e., choosing phases and powering lights). What is changed is the method for which the controller chooses what phase is optimal for the given situation, which is influenced by a neural network. Signal interrupts for pedestrians have not been tested yet, but are discussed in future research (section 6).

1.3. Deep Learning and Genetic Algorithms

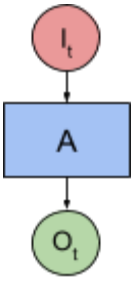
Figure 1.3.1: *Deep neural networks (16)*



In the field of computer science and engineering, often times a scenario cannot be solved for or simulated by hard-coding. For instance, it is relatively easy for a computer to do mathematical computations, but to ask it to identify an object in a photograph is different. A human might easily be able to recognize the object, but not understand how their brain interprets signals given. More complicated models can be created, but there is a limit to what a human programmer could reasonably accomplish. As models grow more and more complex, the ability for programmers to understand every part of a model diminishes, which is why machine learning is used. Machine learning is a broad computational topic in which, generally, computers can be seen to "learn" abilities by being trained on datasets or simulated datasets (7). One system used to accomplish this are neural networks. A neural network is a network of computations used to imitate the complexities of the human brain. Typically, there is an input layer, hidden layers, and an output layer. Neural networks are often used in applications which are too complex to be programmed algorithmically, and are one of the basic structures of machine learning.

Using neuroevolution we can simulate darwinian evolution and natural selection by evolving brains through generations with random mutations (14). A randomly initialized population is generated in a simulated environment. The agents interact with the environment and get a fitness score which corresponds with how well they did at the task (which is specified by the programmer). The agents with higher fitness scores are more likely to be picked for the next generation, and that is determined by a genetic algorithm. Slightly mutated versions of the best agents in the first generation go into the second generation. This generation interacts with the environment and gets a fitness score, then natural selection repeats and theoretically the generations will get higher fitness scores as time progresses.

Figure 1.3.2: *Simple feedforward algorithm (7)*

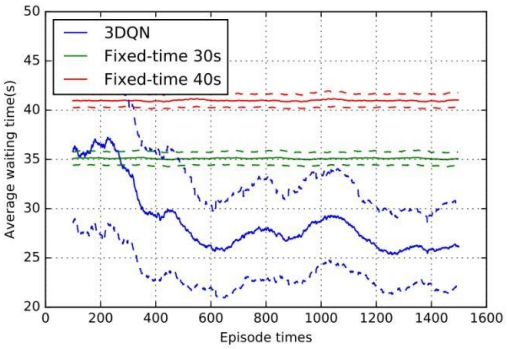


In this study a feedforward neural network was used. In it, an input **I** is being passed through algorithm **A** to produce an output **O** at time **t**. This type of network works in one direction, and is the most basic network form. Though simple in its approach, by adding more hidden layers a more complex situation can be solved for that does not require the use of a different type of network.

1.4. Current State of Adaptive Control Infrastructure and Recent Advancements

Since the problem of traffic exists around the world, it is a popular topic to optimize strategies for reducing traffic. This study focuses on the intersection, so that is where the current state of technology will be shown.

KLD Companies devised a system in 2012 for reducing traffic in midtown New York City (10). They have seen reductions in wait times up to 10% as compared to before it was implemented. The University of Toronto developed MARLIN-ATSC (Multi-agent Reinforcement Learning for Integrated Network of Adaptive Traffic Signal Controllers) in 2014, which has shown to reduce intersection delays almost 40%.
Figure 1.4.1: *Liang et. al. (2018), wait times with deep learning algorithm vs. constant*



Liang et. al. (2018) used a convolutional neural network consisting of an input layer of 60x60x2, which took in all the positions and speeds of vehicles at the intersection (3). The figure shown from the study indicates that the wait times went from 35 seconds for a fixed 30 second light to approximately 25-30 seconds, or a 25% reduction. The current state of research and implementation stands with some working infrastructure changes, albeit at a cost, and also work done in a simulation. This study was completed in simulation as well. The difference is that the code and results open source and have possibly the ability to be implemented for a smaller cost since the computations can all be done on a microcontroller.

2. Materials and Methods

Hardware:

- Raspberry Pi 4, Dell XPS 13 2016
- Electrical Components: Cobbler, breadboards, wires, LEDs, resistors

Software:

- Python 3 and the libraries it comes with
- py5, a graphics library for Python created by Simon Socolow https://github.com/ssocolow/py5
- Neural Network library created by Simon Socolow https://github.com/ssocolow/Neural-Network-Python-Lib

2.1. Modeled Intersection

The intersection chosen to model is the intersection between Broadway, Stillwater, and North Park Street in Bangor, Maine. The reason this intersection was chosen is twofold. Firstly, it is not connected in a chain of intersections. Several roadways in Bangor are set up so that a series of green lights are activated along the path to ensure maximal movement through the given area, especially in more congested areas. Since this particular intersection is relatively isolated, it provides the best test scenario for the traffic

situation that has been modeled. In addition, this intersection provides almost all important aspects of a normal intersection, including four directions, multiple lanes in each direction, and protected left turns, without being needlessly complex. The complexity can be accounted for in further study, however it is not necessary to model in this proof of concept.

Figure 2.1.1: *Modeled intersection facing south on Broadway (via Google Street View)*

The intersection as it currently stands uses cameras as a detection method. There are five cameras installed, four in each roadway and one looking at the left turn lane on Broadway south. These cameras use region of interest (ROI) identification, which means that the traffic engineer sets up a region in the frame, and the detection of a vehicle in that region will trigger the camera to send a signal to the traffic management box (in the upper left corner of Figure 2.1.1., shown in the diagram below). Using this method means that the system only knows if a car occupies the lane, not how many.

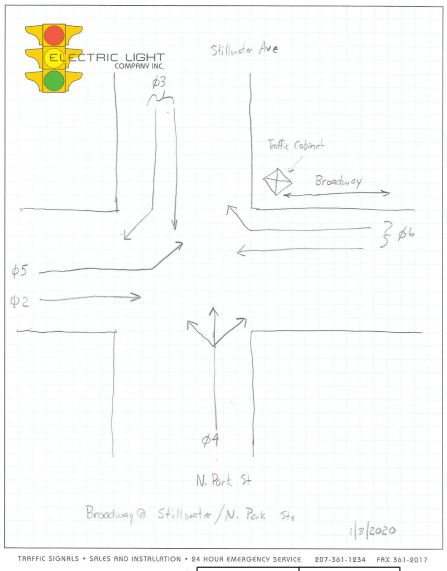


Figure 2.1.2: *Phasing diagram provided by Jeff LaPointe (traffic engineer at the Bangor Public Works)*

The phasing diagram shown to the left indicates the direction of movement for a vehicle at each roadway and lane, as well as the lights that control that particular lane. As stated before, this intersection is influenced by cameras that detect the first vehicle in a lane, and the lights stay on for a set period of time after that first vehicle is detected. The traffic management system then looks for the next camera that has been activated and is not in the current phase. Data for car counts were gathered at this intersection on paper for each of the inbound and outbound lanes in five minute time periods. The results are shown in section 3.1.

2.2. Implementation into Simulation

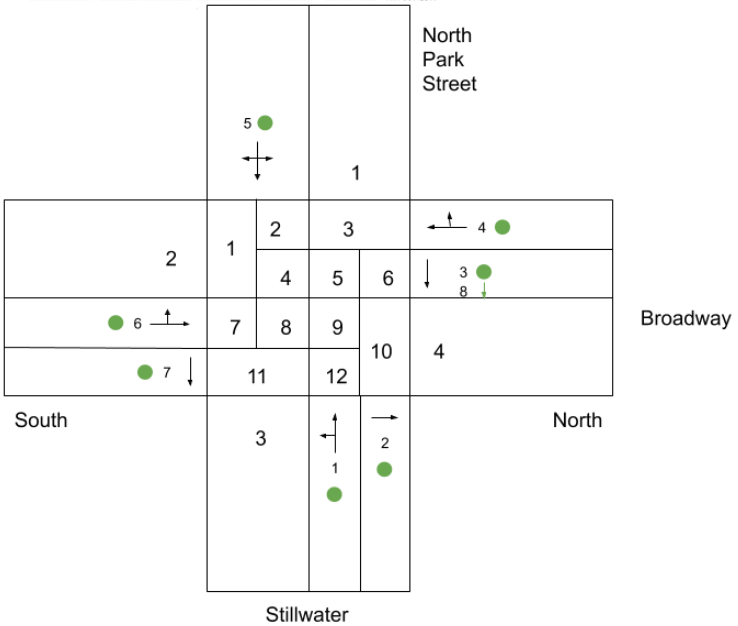


Figure 2.2.1: *Remodeled phasing diagram used in simulation*

The diagram to the left shows the revised phasing diagram, including placement of in-intersection points where vehicles can exist. In addition, each of the inbound and outbound lanes have ten slots, and a

vehicle may spawn in any of the inbound lanes depending on the relative probabilities found by the in-the-field data. This is discussed further in section 2.3.

Several notable changes were made between the simulated intersection and the real one. Firstly, the lights were renumbered for ease of implementation. This change includes right turn lights that act independently from the straightaway lights, which is not how the real intersection works (however it could be rewired to do so) so that they could be controlled independently, and that lights 3 and 8 are modeled separately (yielding left turn vs. protected left turn).

Figure 2.2.2: *Lights activated for each phase*

Phase	Lights Turned On
1	1,2,7
2	2,3,4,8
3	5
4	6,7
5	3,4,6,7

The lights turned on in each phase above are a mix of non-conflicting phases as well as one yielding phase. Phases 1, 2, 3, and 4 all require no yielding. Phase 5 requires a double yield at points 9 and 4 in the in-intersection matrix, however the probabilities for a vehicle going into away lane 1 are lower than a vehicle going straight out of incoming lane 6 (which is shown in figure 3.1.2.). Though the probability is low, there is a chance of gridlock caused by a double yield. Phases 2 and 4 alleviate this by clearing those lanes without conflict. In this study the neural network controls the phase instead of individual lights, which ensures that it does not produce solutions that lead to a conflicting arrangement and a possible crash.

2.3. Creating a Simulation

The simulation was created in the programming language Python, Python3 specifically because it is the newer version. The code can be found to look at or download for free at https://github.com/ssocolow/trafficsim in the spirit of open source.

The simulation of the intersection works by combining three distinct classes. These are the Car class, the Lane class, and the Intersection class. These classes provide the functionality required to make an intersection.

The Car class is a blueprint for what a car is in the simulation. When a car is spawned, it is put into a certain toward intersection lane with a certain target away lane based on the data gathered in the field. For example, a car could have a 20% chance of spawning in toward lane 1 and if it spawns in toward lane 1, an 80% chance to turn left and enter away lane 2 (see figure 3.1.1. and 3.1.2.). Once the car knows where it is and where it is going, it can use this knowledge to decide on a path through the inside of the intersection. For example, this car spawning in toward lane 1 and turning left to enter away lane 2 would have a path through the intersection where it goes through the in-intersection spots 12, 9, 5, 4, 1 (see figure 2.2.1). Cars also have functionality to keep track of how long they have been waiting at the intersection and know where they are inside of the intersection.

The Lane class is a blueprint for what a lane is in the simulation. When a lane is created, it is given a length and a direction. Length is how many spots for cars there are in the lane and direction is if the lane

is going toward the intersection or away from the intersection. The length of all lanes in this simulation is 10, and if a lane has a direction of 1 then it is a toward intersection lane. If it has a direction of -1 then it is an away from intersection lane. Lanes also keep track of their total wait time which equals the sum of each car's wait time in the lane during each time step. Lanes have moving functionality which can move cars through themselves. Each lane stores what is inside of it in a python list called self.contents where zeros mark empty spaces and Car objects mark cars. A lane's contents might look like this if it has a length of 10 [0,Car,0,0,0,Car,0,Car,Car,Car].

The Intersection class is a blueprint which controls the entire intersection. This class contains multiple lanes which control multiple cars. The Intersection class has three important parts: toward lanes, away lanes, and in intersection cars. The toward_lanes (as it is named in the program) store all seven of the lanes going toward the intersection in the case of the modelled intersection. The away_lanes store all of the four away lanes in the intersection model. The in intersection cars (referred to as iic in the program) stores all of the cars inside of the intersection. This class has control over the lights and phases which dictate the groups of lanes that have green or red lights. The most important function in this class is the move function which moves the intersection forward one time step. One time step in the simulation equals one second in the real intersection. The move function combines most of the other functionality of the intersection. Here is a list of what the move function does in chronological order:

1. Increase the wait time counters of the lanes, cars, and intersection.
2. Move away cars from intersection lanes.
3. Move the cars that are in the intersection.
4. Move the lanes that have green lights.
5. If a car is in a lane with a red light and can make a right on red safely, it makes it
6. The lanes with red lights are moved forward but stop and pile up before entering the intersection.
7. Cleanup which contains mapping the in_intersection model for visualization and keeping track of counters which control phase changing.

The cars furthest along their path through the intersection move first to make room for the incoming cars. If the changePhase function is activated, all lanes have red lights for 6 time steps and then the phase is set to whatever was input. 6 time steps model the 4 time steps yellow and the 2 time steps red which is how the real intersection functions. The frequency of which cars are spawned was determined using the collected data (Figure 3.1.3). Assuming the lights are operating during the day, a car spawns once every 2 seconds, or 0.5 cars a second. This could be changed to simulate more or less busy conditions. For now, the code only supports this certain type of intersection, but future work could be done to develop a system that would make it easier to model different intersection types.

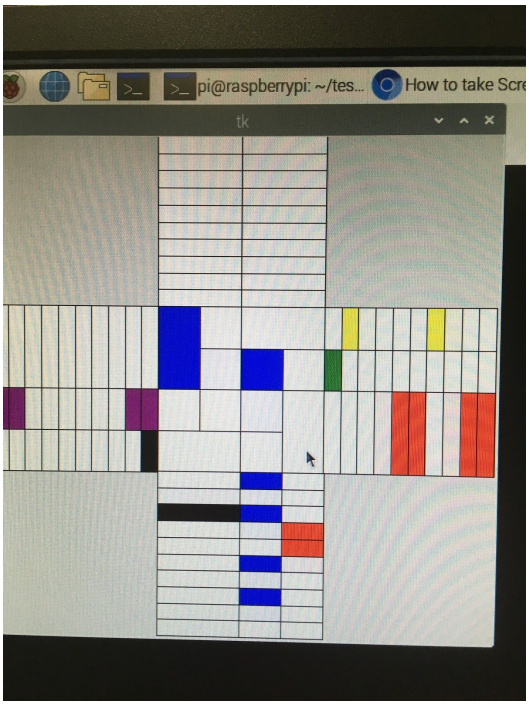


Figure 2.3.1: *Visualization of the intersection on a Raspberry Pi*

A visual representation of the simulation can be seen in the image to the left. The lanes are oriented per the convention demonstrated in Figure 2.2.1. The white spots are where no vehicles are and the colored spots represent a vehicle; the

color determining which lane it originated from. Though useful as a visualization tool, this visual was turned off for all subsequent tests in order to conserve computational time and memory.

2.4. Application of Deep Neural Networks and Neuroevolution

The code can be found to look at or download for free at https://github.com/ssocolow/Neural-Network-Python-Lib in the spirit of open source.

Different versions of the neuroevolution and simulation code were run that had differences in inputs, outputs, population size, epochs, architecture, mutation rate, fitness function, and pool selection algorithms.

The purpose of building the simulation of the intersection was to provide an environment for the deep neural network agents to interact to gain a fitness score. The agents control the phases of the intersection, and after 200 time steps their fitness is determined by how well they minimized wait time or increased throughput. To get the fitness function in a state where higher is better in the case of wait time, we take 1 / the wait time, or the reciprocal.

To begin the neuroevolution process, we first spawn 100 randomly initialized agents. All of the agents go through 200 time steps in the simulation while controlling the phases/lights. At the end of the 200 time steps, the fitness score is derived either from the wait time or throughput. After all the 100 agents go through the simulation and get a fitness score, a probability of being selected from the total pool is made for each agent by calculating the agent's fitness score / the total summed fitness score of all the agents. Then, to populate the next generation, agents are randomly chosen based on their probabilities and mutated versions of themselves go into the next generation. Mutation works by randomly adding a small number between 1 and -1 to the weights and biases of the neural network if a random number is below a certain threshold called the mutation rate.

Then, this generation goes through the process again and the cycle continues. Theoretically, the agents should get higher fitness scores over time because the ones with the higher fitness scores have a better chance to go into the next generation, like Darwinian evolution. The networks were trained for 200 generations in the results section.

The neural network that was tested had a 10x16x16x16x5 architecture. This means there are 10 inputs, a 16x16x16 deep layer, and 5 outputs. The 10 inputs were:

7 from the car count of each lane
1 for how long the signal has been on the current phase
1 for if the lights are red or not
1 for the current phase

The 5 outputs were 1 for each phase. If the neural network decides to switch phase, it will have to go through the 6 second yellow/red sequence. If it chooses the same phase the transition is seamless. The neural network only selects a phase every 10 seconds because this number is around the minimum to which a lane can be cleared. Any less was shown to be significantly slower (figure 3.2.1), and thus was not modeled henceforth.

The mutation rate for the genetic algorithm was variable, and started at 0.2 for generations 1-30, 0.1 for 31-60, and 0.05 onwards. The reason this was implemented was because, since the initial generations are essentially random, we alter the biases more heavily. As the preferred network becomes more refined, the change per generation is allowed to diminish.

Different architectures, population sizes, and mutation rates can be explored and could be part of future research.

2.5. Connecting Software and Hardware

Figure 2.5.1: *Eagle Model 2070 Controller (21)*

Software must be able to interface with the hardware at an intersection in order to be viable as an improvement on the current infrastructure. The controller shown in Figure 2.5.1. (without accounting for the other equipment needed) retails for around ~$200, however it is an older device (21). Newer devices with more modern technology, in addition to the necessary detectors, cost thousands. Besides detection devices, other necessary hardware includes power conduits, backup boards, grounding, cabinet, and physical mounting (21). The Raspberry Pi Model 4 that was used in this study retails at $35 (20). Though this is cheaper than most traffic controllers, the disadvantage is that there has to be multiple adaptations in order for it to work with the current infrastructure. One of those is that the output voltage must be stepped up to 36 volts, which is what is needed to power a traffic light. That means that an external power source must be used, with the Raspberry Pi acting as an interrupt to that source. Also, it must be shielded from the elements, and, whereas the existing box-style controllers have built-in protection, some form of insulation must be used to keep the microcomputer operational. This can be done inexpensively through the use of 3D printing.

However, the Raspberry Pi does offer advantages, which is why it was used in this study. Firstly, it is more versatile than the computers built to manage traffic currently. This means the introduction of new software and neural networks is a possibility, and the system may be updated constantly. In addition, it can be networked wirelessly, which can make it easier to link intersections together. This will be investigated in future research.

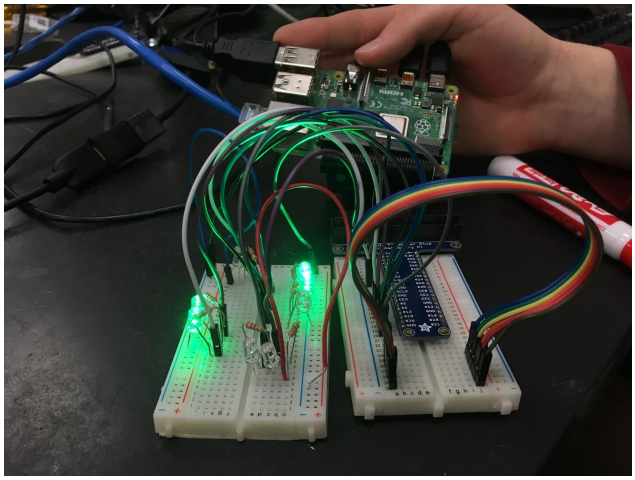


Figure 2.5.2: *Raspberry Pi with LEDs showing traffic light state*

The Raspberry Pi features general purpose input/output (GPIO) ports, which can be used to connect electrical devices to the computer. These were used as a simple proof of concept to show that

the software can be connected to external hardware, i.e. traffic lights. The LEDs in the figure model lights 1-8 and show the current phase activated in the simulation.

3. Results

3.1. Traffic Data

Figure 3.1.1: *Inbound/Outbound lane car counts and relative percent*

Inbound Lane	Car Count	Relative %	Outbound	Car Count	Relative %
1	32	23.5	1	6	7.5
2	19	14	2	23	29
3	15	11	3	18	23
4	27	20	4	32	40.5
5	4	3			
6	29	21			
7	10	7.5			
Total	136	100%	Total	79	100%

Both datasets for the inbound/outbound counts were taken over a five minute time frame on the dates of 12/19/2019 at 3:45 P.M. and 12/28/2019 at 7:42 P.M. respectively. The lane numbering follows the convention as in Figure 2.2.1. The inbound lanes towards the intersection are numbered 1-7 and the outbound lanes are 1-4. Each inbound lane is isolated, however multiple inbound lanes can feed into one outbound. The relative percentages were used to estimate the likelihood for a vehicle to spawn at each location.

Figure 3.1.2: *Percent chance for a vehicle at each inbound lane to feed into their respective outbound*

Inbound Lane	Possible Outbound Lanes	Percent Chance of Arriving
1	1	20
	2	80
2	4	100
3	3	100
4	1	20
	2	80
5	2	31.5
	3	24.5
	4	44
6	1	16
	4	84
7	3	100

Based on the data shown in Figure 3.1.1 an estimation was made for the probability that a vehicle coming from each inbound lane will go to a certain outbound lane. Note that lanes 2, 3, and 7 must turn into their respective right/left only lanes, which is why the probability is 100%.

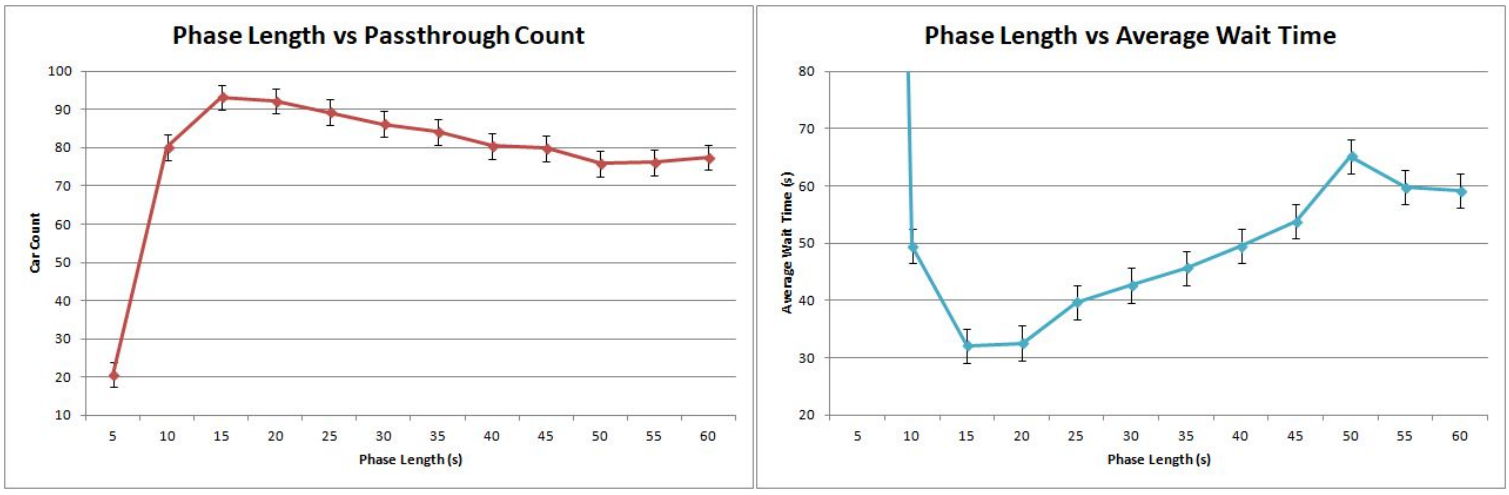
Figure 3.1.3: *Cars per second*

	Inbound Data (12/19/19)	Outbound Data (12/28/19)	AADT 2017 Data (26)
Total Cars	136	79	33308
Timeframe	300 seconds	300 seconds	24 hours (86400s)
Cars per Second	0.45	0.13	0.39

The data for how many cars arrive at the intersection per second can be shown by dividing the number of cars counted by the time frame. The AADT, or average annual daily traffic volume, is data collected by the Maine Department of Transportation for each intersection, the most recent set being taken in 2017. The inbound data was taken when the intersection was relatively saturated at 3:45 P.M., and the outbound data was taken when it was less populated at 7:42 P.M.. The number of cars per second used in the simulation was rounded up to 0.5. This was the case for two reasons. First of all, it is easier to work in increments that can be approximated as whole numbers over a larger time scale. Second of all, rounding up simulates a worst-case scenario, and trains the neural network to accommodate larger traffic volumes if the situation arose (see section 2.3 for implementation into simulation).

3.2. Fixed Time Signals

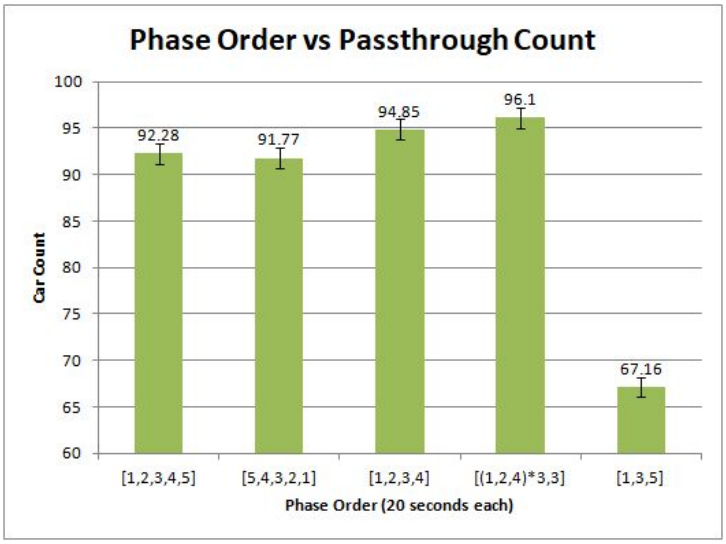
Figure 3.2.1: *Fixed-time phase length vs car passthrough count and average wait time*

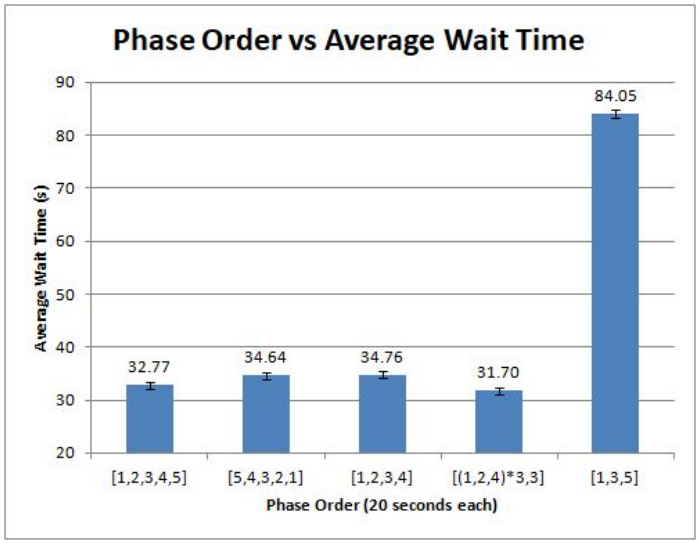


N = 100 for each phase length timing, and plotted +/- 2 standard errors (max error of 3.23 and 1.0 respectively). Each test used the phase order [1,2,3,4,5] (see figure 2.2.1. and 2.2.2) which looped back to the start after it has gone to 5, and a set cutoff time of 200 time steps (s). The only variable changing was the length of time each phase was activated for. The average wait time is calculated by taking the total intersectional time and dividing it by the car passthrough count. The 6 second yellow/red buffer between phases hinders progress of shorter phase length runs as it is a larger fraction of the overall allotted time.

The 5 second phase length is off the chart on the average wait time graph, but the real value is 493.5s. The optimal timing according to this test is 15 seconds, though it is within the margin of error with 20s.

Figure 3.2.2: *Phase order vs passthrough count and average wait time*





N = 100 for each phase order, and plotted +/- 2 standard errors (max error of 1.13 and 0.7 respectively). Each test used a 20 second phase timing, so the only variable changing was the order in which the phases were run (the last phase loops back to the first one), and a 200 second cutoff. Phase 5 is a redundancy of 2 and 4 with left turn yielding, and phases 1-4 are all no-conflict no-yield phases. Phases 1-4 are all that are needed to clear the intersection. The goal was to see if it might be faster to run 5 rather than 2 and 4 separately, but as shown in the [1,3,5] staging, this is not the case. [(1,2,4)*3,3] is the same as [1,2,4,1,2,4,1,2,4,3]. The reason this was tested was because 3 has a much lower percent chance of a car spawning there, so it might be more beneficial to not have 3 run on every cycle. That phasing is within the margin of error with [1,2,3,4], and faster than any test with phase 5 included.

A test with N = 500, phasing [(1,2,4)*3,3], and a phase timing of 15 seconds was conducted to simulate the best results of both of these tests. The average passthrough car count was 94.79, standard deviation of 5.9. The average wait time was 34.94s, standard deviation of 3.2.

3.3. Neural Network Training

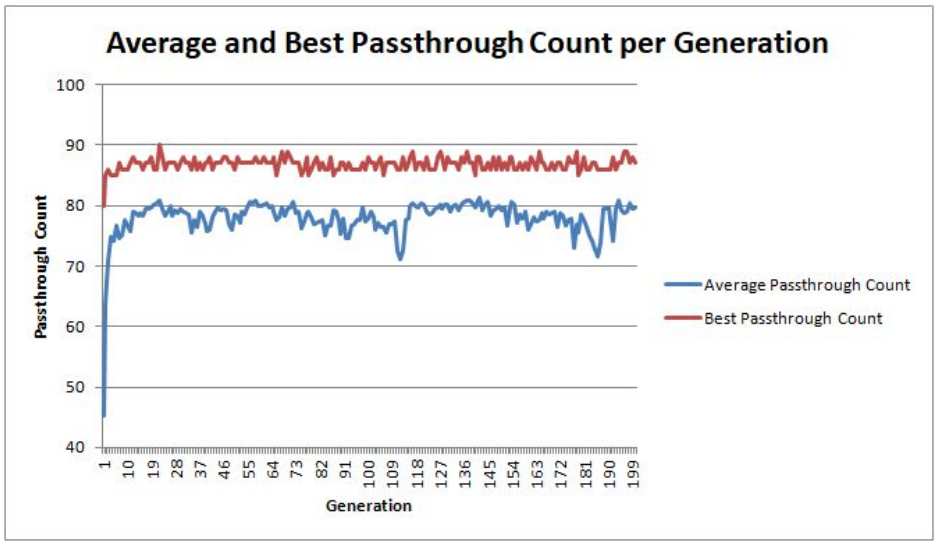


Figure 3.3.1: *Average and best passthrough counts by generation*

Computational time = 625.58s. The best passthrough count remains high, averaging 86.82. This number isn't necessarily the best representation of how the network itself is performing, as random coincidences in car spawning could lead to one

network outperforming the others, but it is not because of the viability of the network. The average passthrough count shows the average of all 100 networks per generation, and is a more accurate representation of the state of training. In this case, the model learned very quickly, jumping from 45 to between 70 and 80 for the rest of the generations.

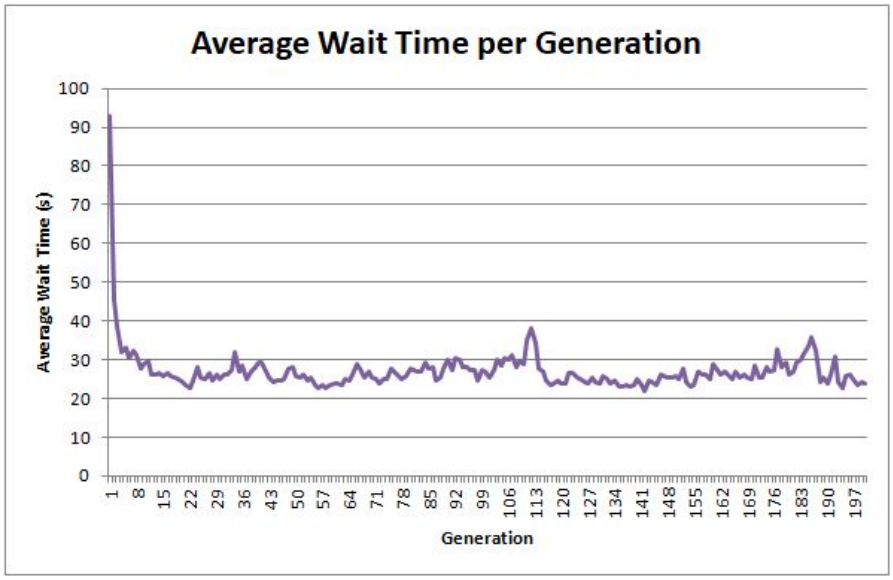
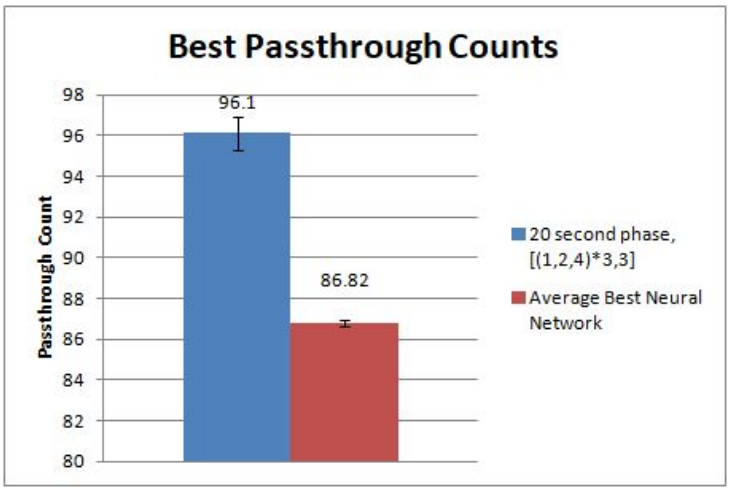


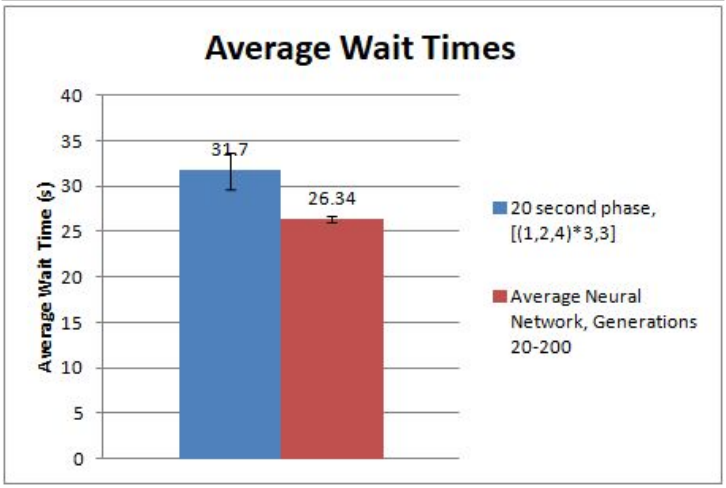
Figure 3.1.2: *Average wait time per generation*

The average wait time per vehicle across all networks started at 93 seconds in the first generation, and then quickly afterwards reduced to between 20 and 40.

3.4. Fixed Time vs. Neural Network

Figure 3.4.1: *Best passthrough counts, best fixed time and neural network*





The staging with the greatest success in the fixed-time light study was the 20 second [(1,2,4)*3,3] model. This was compared to the average best of each generation as shown in Figure 3.3.1. The neural network average for the vehicle throughput is approximately 10% less than with the fixed timing. This will be discussed in section 4.

Figure 3.4.2: *Average wait times, best fixed time and neural network*

The average wait time was the best for the 20 second [(1,2,4)*3,3] model again. The average wait time for the neural networks was derived from the average of generations 20-200 in Figure 3.1.2., as the network plateaued after about 20 generations. The neural networks have an average wait time approximately 17% lower than the best fixed time light.

4. Discussion

The goal of this portion of the study was to confirm that a neural network which controls phasing at an intersection can perform better than a fixed-time counterpart. The reasoning behind the data will therefore be presented in this section.

Results section 3.1 shows traffic data which was collected at the actual intersection. Though not necessarily representative of the intersection at all times of day, the goal was to calculate the relative probability of a vehicle occurrence at each of the inbound and outbound lanes. Assuming an equal distribution of cars at this intersection would have been incorrect, and would be incorrect for the majority of intersections. This data was then fed into a simulation that approximates vehicle movement and the behavior of cars at an intersection. Each run through of the simulation steps through 200 time steps at a rate of 0.5 cars/second. This is on the high end of the cars per second found in the field and using public data, again to simulate a larger saturation of vehicles. If the 200 time steps (seconds) are extrapolated to the 300 seconds that each of the in the field data sets were gathered, the car count can simply be multiplied by 3/2. Doing this for an arbitrary value of 90 arrives at 135, which is close to the throughput seen in the real intersection, confirming the validity of the simulation.

As explained in section 2.1, the real intersection uses cameras as detection into a simple algorithm used to switch the phase. This is called the first-come-first-serve method, or that the first car to arrive in the camera's designated region receives priority - no matter how many vehicles show up afterwards in other lanes. This method specifically was not tested to compare against the neural network, though for saturated intersections it can be assumed that it is similar to fixed-time as a vehicle will almost always be in the camera's ROI. The fixed-time collection was split in order to find the best and worst case scenarios for fixed time phasing. This meant first finding how the throughput and average wait time reacts to a changing in each phase length, and subsequently how they react to phase order. Having too short a phase length, shown in Figure 3.2.1., does not allow adequate time for a vehicle to move across the intersection. The longest path across the intersection is a left turn consisting of 6 spots, or the equivalent of 6 seconds, so anything less than that would run into issue. Average wait times and throughput also diminishes with a phase time too high, where vehicles have to wait for an unnecessarily large amount of time when they are given a red. Changes in phasing as shown in Figure 3.2.2. were done to see if the order and relative frequency of the phases mattered, and for this particular intersection it does. Phase 3 only clears the least populated lane, so when it is run less than once per stage, then the model as a whole runs faster. Of course, there is a limit to the improvements one can make on fixed-time staging, which is why the best of these tests were used to compare to the neural networks.

Section 3.3 shows how the networks were trained over 200 generations. The best improvement occurs in the first few generations and the returns diminish soon afterwards. In some cases the generation as a whole performs worse than the previous generation, but that is because of circumstance and because the biases are mutated by the mutation rate, which can go either way.

Figures 3.4.1. and 3.4.2. appear to conflict, showing that the network both had a lower average throughput count and a lower average wait time. Though it is impossible to find the exact cause with the data extracted, a hypothesis can still be formed. The speculated reason is that lower average wait time does not

necessarily correlate to having a higher throughput. The total intersectional wait time is a combination of each of the vehicle's wait times. A lane of vehicles that has been waiting for a long time will compound the wait times to a greater extent than a free flowing one that has to wait very little. Therefore, clearing each lane because of a single vehicle waiting a long time is preferential than perhaps making a more populated lane constantly flowing. A constantly flowing lane for the whole time interval might end up with a higher throughput, but at the expense of lanes that are going to be compounding their wait times the whole time. Either way, the network on average performed 17% better than the best fixed time staging, which is an improvement that can be increased with future optimization.

The evolutionary process was run multiple times, but only 2 of the times it produced results that were better than the fixed time lights. The networks are all initialized randomly, so it is theorized that some of the ones that did the best in the first generation could be from random luck of choosing the right phase and never changing (because the networks didn't usually change phases). Then, once their DNA was used to spawn some of the next generation, the next generation wouldn't get the same luck so would have worse throughput and more wait time. The major problem is that the neural networks don't switch phases often and usually keep one phase the entire simulation time. In order to get a functional neural network that does not just have a higher fitness score because of initial luck, the evolution was run until there was a favorable starting condition for at least 1 network out of the 100 initialized so that it could continue evolving preferably. This could be an error in the simulation, the fitness function, the feed-forward neural network architecture, or simply the complexity of the situation. Whatever the reason, it must be investigated further in order to have this algorithm be applicable to more situations.

5. Conclusions

1. Neural networks were trained to control a simulated traffic signal using a genetic algorithm in order to reduce average wait times at an intersection 17% as compared to the best fixed-time staging that could be made.
2. Reducing average wait times per vehicle does not necessarily correlate to the throughput of the intersection.
3. An open source architecture for the simulation, neural network, and genetic algorithm were created, can be run on a Raspberry Pi, and are available for free in order to make the technology more accessible.
4. A feed-forward neural network can be trained relatively quickly (625 seconds), however starting conditions and randomization generally affect the progress of evolution greatly.
5. The best phase length for the intersection studied (and other intersections of generally the same scale) is between 10 and 20 seconds - which is mediated by the amount of time it takes a vehicle to pass through the center of the intersection.

6. Future Work

Several things can be done in order to improve the results of this study and make them more applicable. The neural network made a decent improvement to the simulated wait times, but it is yet to be studied in a real world situation. Other studies that are completely simulated as well have achieved reductions of up to 25% (granted it was compared to a fixed 30 second, which this study showed was not the fastest fixed timing), whereas a real world counterpart might achieve 10%. This reduction is likely going to have to

increase for the network strategy created to be more viable, which can be done through revisions and optimization of the code. These optimizations include testing different mutation rates, population sizes, etc. In addition, the code at the moment only works for the particular intersection studied. In order for this code to reach a wider audience and perhaps be implemented, it needs to be able to be easily developed for different intersections. This necessitates the creation of a graphical user interface in order to piece parts of an intersection together visually, and also train the networks on many different variations of intersections.

An important piece that was not studied was the interaction between simulated reality and reality, which requires detection. As stated, the real intersection uses single-region cameras. Technology that can give the network the same inputs as are fed in simulation must be connected. This technology includes integration with computer vision or solutions using older induction loops, microwave sensors, ultrasonic, and single-region cameras; which would necessitate a retraining of the network based on new given information. Other aspects of the integration of this software were discussed in section 2.5., and the pros and cons of using the Raspberry Pi over existing controllers.

One aspect that many studies overlook is the inclusion of pedestrians into the simulation. Pedestrians act differently than vehicles as they move with different paths, at a different rate, and the walk-signal phases must be set up differently. An input must be created for a pedestrian who pushes the walk-button, and they most likely should be given priority. These are things that have either not been tested or should be changed in order to improve this study.

7. References

1. Flynn, M. R., Kasimov, A. R., Nave, J., Rosales, R. R., & Seibold, B. (2008). Self-sustained nonlinear waves in traffic flow. *Physical Review E,79*(5). doi:10.1103/physreve.79.056113
2. Hadi, R. A., Sulong, G., & George, L. E. (2014). Vehicle Detection and Tracking Techniques : A Concise Review. *Signal & Image Processing : An International Journal,5*(1), 1-12. doi:10.5121/sipij.2013.5101
3. Liang, X., Du, X., Wang, G., & Han, Z. (2018). Deep Reinforcement Learning for Traffic Light Control in Vehicular Networks. *IEEE Transactions on Vehicular Technology*. doi:arXiv:1803.11115v1
4. Serrano, Á, Conde, C., Rodríguez-Aragón, L. J., Montes, R., & Cabello, E. (2005). Computer Vision Application: Real Time Smart Traffic Light. *Lecture Notes in Computer Science Computer Aided Systems Theory – EUROCAST 2005,*525-530. doi:10.1007/11556985_68
5. Younes, M. B., & Boukerche, A. (2014). An Intelligent Traffic Light scheduling algorithm through VANETs. *39th Annual IEEE Conference on Local Computer Networks Workshops*. doi:10.1109/lcnw.2014.6927714
6. Zhang, K., & Batterman, S. (2013). Air pollution and health risks due to vehicle traffic. *Science of The Total Environment,450-451*, 307-316. doi:10.1016/j.scitotenv.2013.01.074
7. Banerjee, S. (2018, May 23). An Introduction to Recurrent Neural Networks – Explore Artificial Intelligence – Medium. Retrieved from https://medium.com/explore-artificial-intelligence/an-introduction-to-recurrent-neural-networks-72c97bf0912
8. Chapter 2, Traffic Detector Handbook: Sensor Technology. (n.d.). Retrieved from https://www.fhwa.dot.gov/publications/research/operations/its/06108/02.cfm
9. INRIX. (n.d.). INRIX Global Traffic Scorecard. Retrieved from http://inrix.com/scorecard/
10. Midtown In Motion. (n.d.). Retrieved from https://www.kldcompanies.com/midtown-in-motion/
11. p5.js. (n.d.). Retrieved from https://p5js.org/
12. Rosebrock, A. (2017, September 18). Real-time object detection with deep learning and OpenCV. Retrieved from https://www.pyimagesearch.com/2017/09/18/real-time-object-detection-with-deep-learning-and-opencv/

13. Razali, N. M., & Geraghty, J. (2010, January). Genetic Algorithms Performance Between Different Selection Strategy in Solving TSP. Retrieved from https://www.researchgate.net/figure/Genetic-algorithm-procedure-for-TSP_fig1_236179246
14. Shiffman, D. (n.d.). Chapter 10. Neural Networks. Retrieved from https://natureofcode.com/book/chapter-10-neural-networks/
15. Clean Energy Challenge 2018. (n.d.). Retrieved from https://cleanenergychallenge.whatdesigncando.com/
16. Nielsen, & A., M. (n.d.). Neural Networks and Deep Learning. Retrieved from http://neuralnetworksanddeeplearning.com/chap5.html
17. Beijing reveals annual vehicle emissions load. (n.d.). Retrieved from https://www.eco-business.com/news/beijing-reveals-annual-vehicle-emissions-load/
18. Greenhouse Gas Emissions from a Typical Passenger Vehicle. (2018, May 10). Retrieved from https://www.epa.gov/greenvehicles/greenhouse-gas-emissions-typical-passenger-vehicle
19. Transport, D. F. (2009, December 31). Traffic advisory leaflets from 1989 to 2009. Retrieved from https://www.gov.uk/government/publications/traffic-advisory-leaflets-1989-to-2009
20. What is a Raspberry Pi? (n.d.). Retrieved from https://www.raspberrypi.org/help/what- is-a-raspberry-pi/
21. "Traffic Control Systems Handbook: Chapter 7. Local Controllers." *Traffic Control Systems Handbook: Chapter 7 Local Controllers - FHWA Office of Operations*, U.S. Department of Transportation, ops.fhwa.dot.gov/publications/fhwahop06006/chapter_7.htm
22. (2019). *Arxiv.org*. Retrieved 3 December 2019, from https://arxiv.org/pdf/1807.01628.pdf
23. *These AI Traffic Lights Could Shorten Your Commute*. (2016). *Popular Mechanics*. Retrieved 3 December 2019, from https://www.popularmechanics.com/technology/infrastructure/a23438/ai-traffic-lights-pittsburgh/
24. Baker, F. (2019). The technology that could end traffic jams. Bbc.com. Retrieved 3 December 2019, from https://www.bbc.com/future/article/20181212-can-artificial-intelligence-end-traffic-jams
25. (2019). *Utne.com*. Retrieved 3 December 2019, from https://www.utne.com/science-and-technology/marlin-transforms-traffic
26. 2017 Maine Transportation Count Book. (2017). Retrieved from https://www.maine.gov/mdot/traffic/docs/ytc/2017/CountReport_Penobscot2017.pdf
27. Baker, F. (2018, December 12). The technology that could end traffic jams. Retrieved from https://www.bbc.com/future/article/20181212-can-artificial-intelligence-end-traffic-jams