Machine Learning Engineer Nanodegree

Capstone Project:

Traffic Lights Planning

TSUNGEN SU
June 4th, 2017

I. Definition

Project Overview

In the “train the smartcab to drive” project, the smartcab learned how to reach its destination safely and efficiently by using Q-learning. Traffic lights, on the other hand, is another important factor for the traffic system. Traffic light control system is pre-programmed by trained traffic engineer and is used to control traffic flow. However, if the traffic light control system can be self-learned for many situations that might be able to reduce traffic congestion and improve traffic efficiency.

The similar research paper “Reinforcement Learning for Adaptive Traffic Signal Control” suggested utilizing reinforcement learning, specifically on Q-Learning, to dynamically optimize a standard four-way intersection stop light. However, it only focus on one four-way intersection.

This project aim at minimising car waiting time by a self-learned traffic light planner in a finite space. For example, the10x10 grid space in a four-way intersection the traffic light control system should stop the direction which has less car in waiting and most cars are allowed to passing. The smartcab project simulator is reused in this project for traffic situation. The all intersection are concerned in the area, not just in one four-way intersection.

Problem Statement

The problem with traditional traffic light control is it is running on regular interval basis regardless traffic situation. It has only two actions which are open ways for north-south or east-west directions. This task is simple, switch them in a time interval. However, to improve traffic flow that requires human intervention. We can say that is not intelligent.

What can be done in order to make it intelligent? By saying intelligent, is to set a goal to find a way that improves the problem. To make it measureable, first thing is to build a model of given problem:

Giving an 8 x 4 grid square area, the line to be thought of roads, the intersection to be thought of 4 way intersection. The input and output are defined as following:

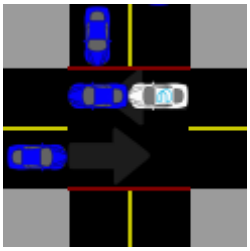


Figure 1 4 way intersection

Input Xi: the number of cars on the road in the given the position. For example, figure 1 has 4 cars on the road, the coordinates are (1,2), (2,2), (2,3) and (3,1). Coordinate starts at left top corner and start from 1. We can see the corners are blocks which no car can drive on.

Output Yi, the action of traffic light is taken in an intersection. 0: Green-NS and Red-EW, 1: Red-NS and Green-EW. We can think of this as possibility of opening North-South way, if it is very low then it is meat to open East-West way. In figure 1, the red line on the north-south direction means close way.

Metrics

We can define performance metrics as a score of running car divided by total car number after that traffic light control system has made a decision. More cars in the running state means that traffic light control system is performing well.

Car state: stall (cannot move), running.

Actions: North-South open way, East-West open way.

Score: (total car on the road – total stall car) / total car on the road.

II. Analysis

Data Exploration

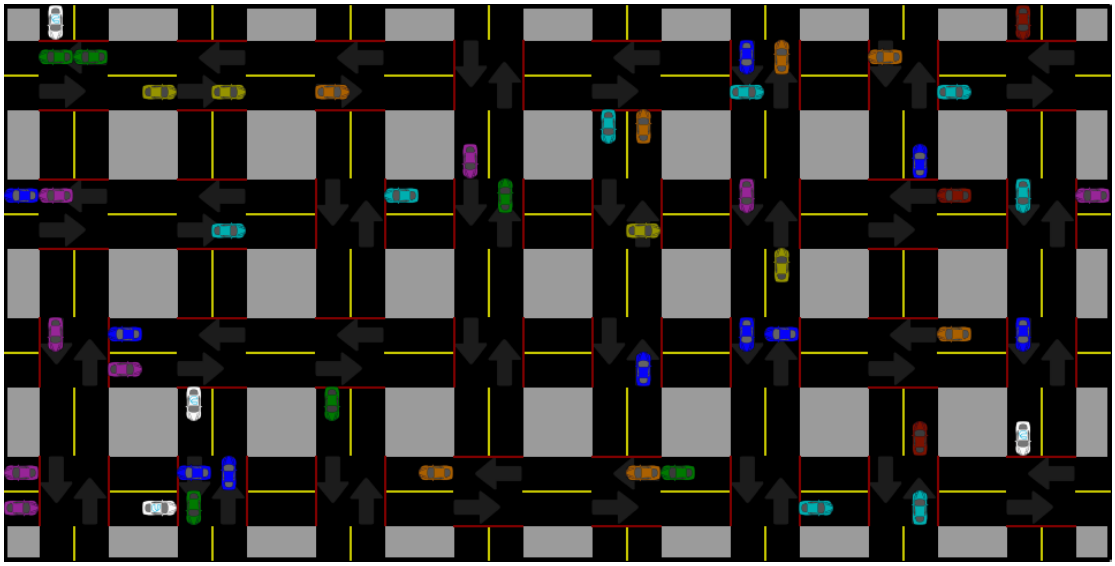


Figure 2 the 8 x 4 environment

In this project, the data is generated from random walking cars. Given 8x4 grid environment (Figure 2), there are 32 four-way intersections, each intersection has 12 spaces that allow car to drive on. There are 384 spaces in total, which will be used as input for the data. Given 1 as there is a on the position and 0 as vacant, there are 2^384 possible outcomes which is very large number and it can be impossible to run all the states or even save in the memory.

There are two actions to an intersection, in the 8 x 4 grid environment there are 32 intersections in total which is 2 ^ 32 combinations of actions that agent can take.

Algorithms and Techniques

The total combinations of all states and actions for 8 x 4 grid environment that would be 2 ^ 384 x 2 ^ 32, which is 2 ^ 416. To convert it to gigabyte, 2 ^ 416 / 2 ^ 30 is 2 ^ 386 gigabyte. It is impossible to save all the information in the memory or even running for checking all possible combinations.

In order to solve this problem is to approximate the best actions for the given state. The algorithm to propose here is to using neural network architecture to estimate the best outcomes. By using neural network, the coordinates is taken as input. The number for input layer would be only 384, and the output would be the action of each intersection, which is 32.

384 input and 32 output is not a big number, and it if very efficient to compute and store in memory.

Benchmark

The fixed switching traffic light control could be the benchmark to compare to our solution model. The defined score is used for comparing the performance of models. Figure 3 shows the benchmark result for 50 and 100 cars running in the 8 x 4 gird environment with 2 trials, each trial is running for 5 minutes. The standard deviation is a small number, which mean the result is very stable, the performance is highly centralized, we will not see the score 90 in first minute and 40 in the last minute.

		score	stall	tick
cars				
50	count	3552.000000	3552.000000	3552.000000
	mean	70.786779	14.701858	889.591216
	std	0.920839	5.483373	512.917123
	min	69.620000	2.000000	2.000000
	25%	70.200000	10.750000	445.750000
	50%	70.860000	14.000000	889.500000
	75%	71.080000	18.000000	1333.250000
	max	90.000000	31.000000	1795.000000
100	count	2732.000000	2732.000000	2732.000000
	mean	62.681570	37.373719	684.500366
	std	0.795333	8.293708	394.402951
	min	59.580000	14.000000	2.000000
	25%	62.580000	31.000000	343.000000
	50%	62.700000	36.000000	684.500000
	75%	62.810000	44.000000	1026.000000
	max	82.000000	62.000000	1368.000000

Figure 3 Benchmark result

III. Methodology

Data Preprocessing

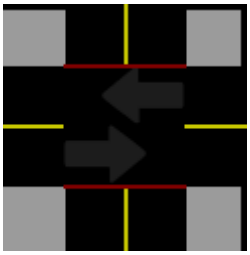
Data is auto generated in real time, it does not require pre-processing beforehand.

Implementation

To simulate the traffic situation, we need to create an environment that is close to our definition. So, some of codes are copied from the "train the smart cab to drive" project. The amendment was made to match our problem definition. Such as coordinates of position to the road and random driving cars. Figure 4 shows the template of creating road. The 8 x 4 grid environment will have 8 x 4 = 32 templates. All templates are arranged in a column by row sequence order. The coordinate is ordered from top left to right bottom starts from 1.

The coordinate indexed dictionary object is created for saving direction and placeholder tuple. The car object is replaced the "None" when there is a car in the position of the road.

loads[(2,1)] = (TS, None)



```
road template :
X |TS     |TN     |X
TW|L(TSW)|L(TNW)|TW
TE|L(TSE)|L(TNE)|TE
X |TS     |TN     |X
```

Figure 4 Road Template

```
TNW = ((0,  -1), (-1,  0)) # Toward North or West, north is two times important
TNE = ((0,  -1), (1,  0)) # Toward North or East, north is two times important
TSW = ((0,  1), (-1,  0)) # Toward South or West
TSE = ((0,  1), (1,  0)) # Toward South or East
TN = ((0,  -1),) # Toward North
TS = ((0, 1),) # Toward South
TE = ((1,  0),) # Toward East
TW = ((-1, 0),) # Toward West
```

Figure 5 Direction Code

The TrafficLightControl interface (Figure 6) is used to control the traffic lights in the environment. The Environment object will use control interface to build a traffic light object, register a traffic object by giving positions.

```
class TrafficLightControl(object):

    def __init__(self):
        self.lightPositions = OrderedDict()
        self.env = None

    def build_light(self):
        pass

    def setup(self):
        pass

    def set_env(self, env):
        self.env = env

    def signal(self):
        pass

    def after_signal(self):
        pass

    def reset(self):
        pass

    def register(self, light, positions):
        for position in positions:
            self.lightPositions[position] = light

    def allow(self, position, heading):
        """already in intersections"""
        if self.lightPositions.has_key(position):
            return True
        """check next position if it is in the intersections"""
        pos = (position[0]+heading[0], position[1]+heading[1])
        if self.lightPositions.has_key(pos):
            light = self.lightPositions[pos]
            if light.get_open_way() == TrafficLight.NS:
                return heading == (0, 1) or heading == (0, -1)
            else:
                return heading == (1, 0) or heading == (-1, 0)
        else:
            return True
```

Figure 6 Traffic Light Control Class

The simulator will trigger environment tick function (Figure 7) for updating the dynamic. When the environment object is ticked, it will send the signal to traffic light control object and then calling step function to all the cars on the road. The step function of car object is used to decide whether it should

moving forward to their direction or stop for the traffic light control or car in front. The after_signal function allowed traffic light control object to see the result after action taken.

```
def tick(self):
    self.t += 1
    self.stall = 0
    self.control.signal()
    self.number_of_car = 0
    for pos, t in self.roads.items():
        pos, obj = t
        if type(obj) is Car:
            self.number_of_car += 1
            obj.step()
            if obj.state == 0:
                self.stall += 1
    self.totalStall += self.stall
    self.control.after_signal()
```

Figure 7 tick function of environment object

The simple neural network model is implemented for the use of this project. Figure 6 shows the class diagram of created Neural Network classes. The linear, relu and sigmoid activation functions are implemented. The quadratic and cross entropy cost functions are implemented.

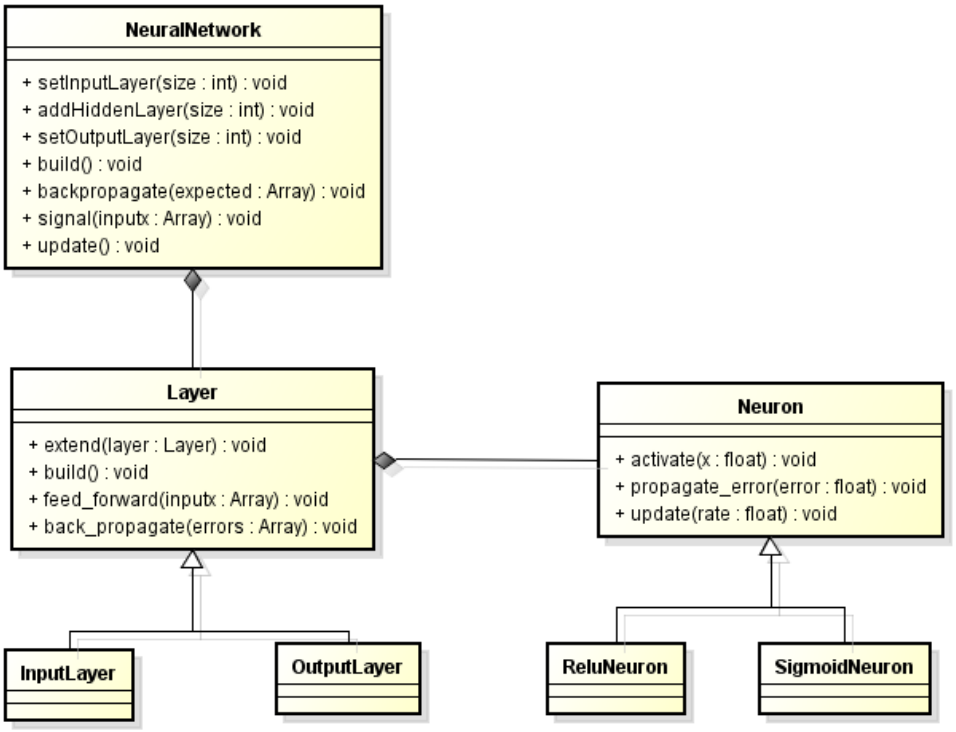


Figure 8 Neural Network Class Diagram

```
def setup(self):
    self.model = NeuralNetwork(alpha=self.alpha)
    self.model.set_input_layer(len(self.env.roads))
    self.model.add_hidden_layer(len(self.env.roads), activation="linear")
    self.model.set_output_layer(len(self.lights), activation="sigmoid")
    self.model.build()
    self.model
    for i in range(0, len(self.lights)):
        self.lights[i].neuron = self.model.output_layer.neurons[i]
```

Figure 9 setup neural network

Figure 8 shows the architecture for this project. One hidden layer with same amount of input layer's neurons is added.

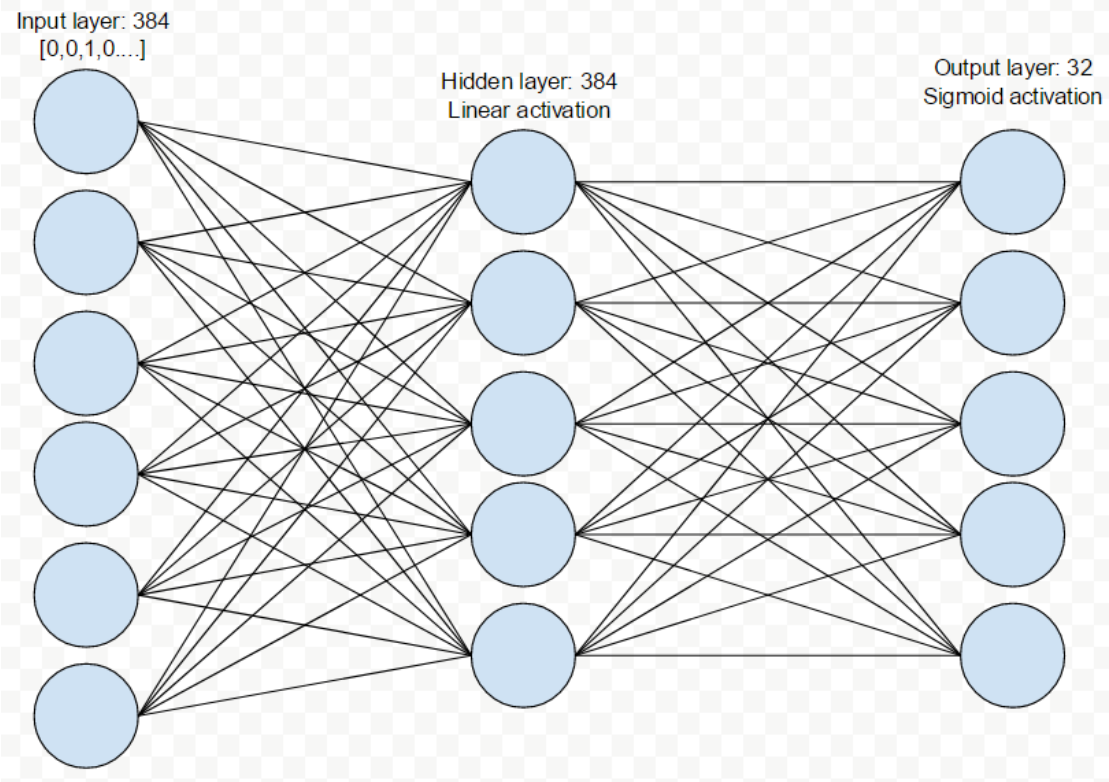


Figure 8 Neural Network Agent Architecture

At output layer, the sigmoid activation is used for controlling the traffic light at each intersection. To justify the correct output values for neural network, the stall car in an intersection is used for voting the correct action to take. The following is defined for the expected values to their correct value:

North-South open way: 1

East-West open way: 0

The quadratic cost function rather than cross entropy is because there is case where either action is acceptable. In this case we use 0.5 to denote any action. So when ever that are equal vote, the 0.5 is used as expected value. Figure 10

shows the neuron traffic light object defines the threshold for traffic light control. If the value is between 0.4 and 0.6, then random action is taken. This will allow preventing overfitting problem.

```
class NeuronTrafficLight(TrafficLight):

    def __init__(self, open_way=None):
        super(NeuronTrafficLight, self).__init__(open_way)
        self.neuron = None
        self.ns_vote = 0
        self.ew_vote = 0

    def switch(self):
        """if neuron output is greater than 0.5 then opens North-South way"""
        if self.neuron.output < 0.4:
            self.open_way = self.EW
        elif self.neuron.output > 0.6:
            self.open_way = self.NS
        else:
            self.open_way = random.choice([self.EW, self.NS])
```

Figure 10 Neuron Traffic Light class

Refinement

There are a few parameters that can be tune when using neural network as solution, such as learning rate or the number of hidden layers with different activations.

The learning rate improves the overall score when it is low, the 0.15 is the number that gives us the best score at training stage.

The more hidden layer doesn't improve the model performance, the one hidden layer with 384 neurons tend to present best result.

IV. Results

Model Evaluation and Validation

Figure 11 shows the training score of 50 cars on the road for 20 minutes. The same model is used to train 100 cars on the road in 20 minutes, see figure 12. It is clear that more cars on the road affects the performance dramatically. The training result is shown on Figure 13.

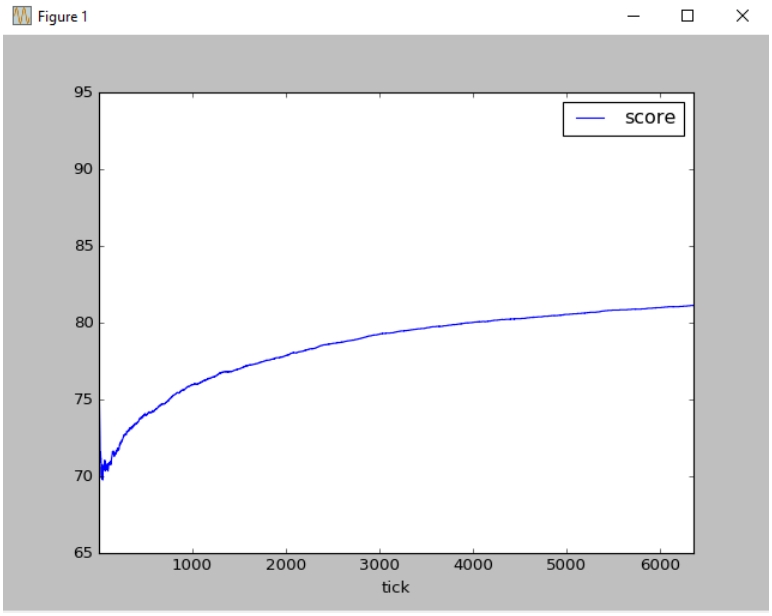


Figure 11 50 cars training

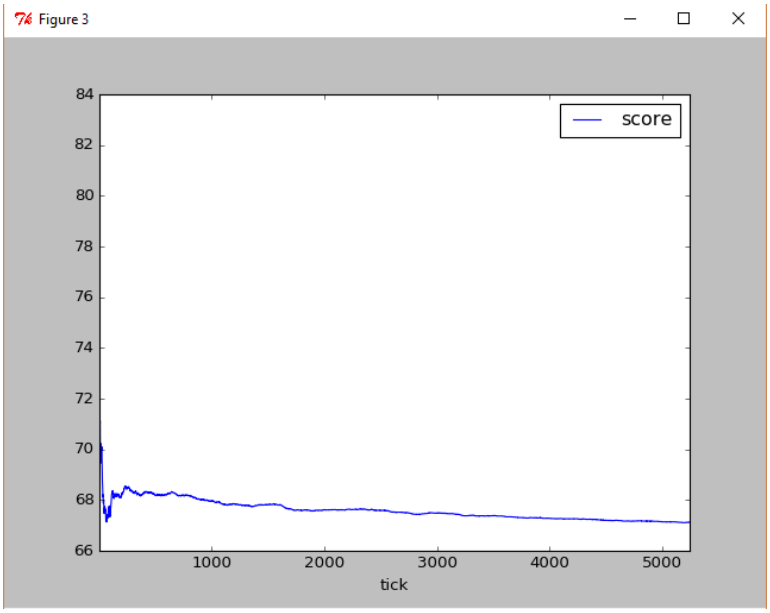


Figure 12 100 cars training

		score	stall	tick
cars				
50	count	13081.000000	13081.000000	13081.000000
	mean	80.052169	9.239966	3274.240654
	std	2.366682	3.102009	1892.459417
	min	69.760000	1.000000	2.000000
	25%	79.480000	7.000000	1637.000000
	50%	80.940000	9.000000	3272.000000
	75%	81.660000	11.000000	4907.000000
	max	97.000000	24.000000	6722.000000
100	count	10495.000000	10495.000000	10495.000000
	mean	67.339041	33.018961	2625.250214
	std	0.570130	4.980362	1514.895287
	min	66.710000	13.000000	2.000000
	25%	66.980000	30.000000	1313.500000
	50%	67.200000	33.000000	2625.000000
	75%	67.590000	36.000000	3937.000000
	max	86.000000	52.000000	5250.000000

Figure 13 Training result

The testing result for 50 cars and 100 cars random walking on the road for 5 minutes is shown on Figure 14. The testing result is very close to training result, it looks good statistically. The standard deviation is a very small number in both 50 cars and 100 cars trail.

		score	stall	tick
cars				
50	count	1763.000000	1763.000000	1763.000000
	mean	80.317005	9.883154	883.000000
	std	0.560597	2.992417	509.078579
	min	79.780000	1.000000	2.000000
	25%	80.140000	8.000000	442.500000
	50%	80.240000	10.000000	883.000000
	75%	80.360000	12.000000	1323.500000
	max	91.000000	20.000000	1764.000000
100	count	1361.000000	1361.000000	1361.000000
	mean	65.544129	34.230713	682.000000
	std	0.850694	5.058159	393.031169
	min	64.750000	20.000000	2.000000
	25%	65.320000	31.000000	342.000000
	50%	65.420000	34.000000	682.000000
	75%	65.600000	38.000000	1022.000000
	max	83.500000	49.000000	1362.000000

Figure 14 Testing result

Justification

The Table 1 shows the score comparison of benchmark and neural network model. The model seems perform well then benchmark model. It is improved 13% on 50 cars and 4.5% on 100 cars.

	50 car mean score	100 car mean score
Benchmark	70.78	62.68
Neural Network Model	80.31	65.54

Table1 Model Comparison

The Neural Network Model improves traffic flow by controlling traffic lights. The interesting thing is it less improved when there are more cars on the road. It can be many reasons, such as the probability of cars congest in an intersection is too high to make traffic flow, or this neural network model work better for 50 cars, the models for 100 cars have not been found. It is possible there are other great neural network models for this problem. There are also many other activation functions have not been tried and implemented in this project.

Using neural network model to approximate the best action in traffic light control is feasible, it improves the traffic performance then traditional model.

V. Conclusion

Reflection

The “train smart cab to drive” project uses reinforcement learning to train the smart cab driving to the destination. This project is a twist version of smart cab, it intended to make traffic flow more fluent. Unlike smart cab project the states and actions are manageable. The number of states and actions is too large, it is impossible to kept in memory or finding a best action in a reasonable time. Using Q-Learning for this problem can be inappropriate.

The neural network model might be a good choice for outcome approximation. It doesn’t have memory problem like Q-Learning need to keep actions per state in memory. The problem is to find the neural network architecture that is best for the problem to solve.

For Q-Learning, it is important to agent that it can experience the most of possible state with all action taken. If both the number of state and action are large it is very difficult to the agent performing well and generalize things. For Neural network model, it is important to have just enough data for training the weights of each layers. It can be time consuming on training, but it works very efficient on testing. For this project, the neural network model is better than Q-Learning for the problem that we are trying to solve.

Improvement

The architecture of neural network that was chosen for this project is possible to be improved. There are many activation functions haven’t been implemented and tried in this project. It is possible there are better model for the problem.

Although neural network model improves the result by approximating the best outcomes of each intersection. This design has its drawback, the expected value of each outcome are short sighted. It only look at situation of local intersection to decide the best action to take, the overall environment is not considered. It is possible that take wrong action at one intersection can improve score afterward.

The further research paper to be look at for this problem is DQN, Deep Q-learning Network. This seems can produce better result by experience replay mechanism. It might be interesting to see how it is combine the Q-Learning and deep learning techniques.