PROJECT

ENPM 808F - ROBOT LEARNING

Deep Reinforcement Learning for Simulated Self Driving Car

Submitted by:

Gunjan Khut [115813083]

ABSTRACT:

Reinforcement learning is considered to be a strong AI paradigm which can be used to teach machines through interaction with the environment and learning from their mistakes. Despite its perceived utility, it has not yet been successfully applied in automotive applications. The mixture of Reinforcement learning with Deep Learning was pointed out to be one of the most promising approaches in achieving human-level control. This has been demonstrated in learning of games like Atari and GO, where a computer has been successfully trained to learn these games. Motivated by this, a demonstration of the self-driving car using deep Q learning is presented here. This is of particular relevance as it is difficult to pose autonomous driving as a supervised learning problem due to strong interactions with the environment including other vehicles, pedestrians, and roadworks.

The project uses a car model developed in Graphical interface and enables it to travel from one point to another in a rectangular map using Deep Q learning. The car also displays collision avoidance property while moving in the simulation environment.

Q learning with tables for storing Q values is not very effective when a number of state increase, hence Deep Q Learning is used in this project. Deep Q-Learning is the result of combining Q-Learning with an Artificial Neural Network. In DQN the states of the environment are encoded by a vector which is passed as input into the Neural Network. Then the Neural Network tries to predict which action should be played, by returning as outputs a Q-value for each of the possible actions. Eventually, the best action to play is chosen by either by overlaying a SoftMax function. In this project, experience replay has been used to improve learning.

Table of Contents

1. INTRODUCTION:

Driving is a multi-agent interaction problem. As a human driver, it is much easier to keep within a lane without any interaction with other cars than to change lanes in heavy traffic. The latter is more difficult because of the inherent uncertainty in the behavior of other drivers. The number of interacting vehicles, their geometric configuration and the behavior of the drivers could have large variability and it is challenging to design a supervised learning data set with an exhaustive coverage of all scenarios. Most of the algorithms used now are only suitable for avoiding stationary obstacles and/or moving obstacles whose future position and moving direction are predictable. In other words, algorithms need maps of its environment to avoid collisions. However, generating a map of the unknown environment is practically impossible.

This project aims to solve the problem of collision avoidance in an unknown environment using reinforcement learning.

2. RELATED WORK

The neural network has been used in the past for autonomous vehicles and has been successfully implemented on a real vehicle. Pomerleau [4] developed ALVINN (Autonomous Land Vehicle in a Neural Network), a 3-layer back-propagation network designed for the task of the road following. ALVINN takes images from a camera and a laser range finder as input and produces as output the direction the vehicle should travel in order to follow the road. Though much of the work involving reinforcement learning has been done on a simulator. Jeffrey Roderick Norman Forbes presented Reinforcement Learning for Autonomous Vehicles as a thesis. April Yu et al. [5] presented Deep Reinforcement Learning for Simulated Autonomous Vehicle Control.

3. APPROACH

3.1 SIMULATION ENVIRONMENT

The first step in this project was to develop a car model and sensor using Kivy and then integrating it with python scripts to add attributes to it. Three sensors will be used to detects the obstacle which is Left, Right and the front of the car. Use of Graphical interface did not affects the project goal as all the parts like a sensor, obstacle, etc. can be easily generated through programming

3.2 MARKOV DECISION PROCESS

As stated, earlier Reinforcement learning is a very good method to teach the agent through interaction with the environment and RL problem can be formalized commonly in Markov Decision Process. Markov decision processes (MDPs) provide a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. A Markov Decision Process is a tuple (S, A, T, r, γ) where: S is a finite set of states, A is a finite set of actions, T is a state transition probability function, r is a reward function, γ is a discount factor.

3.3 Q-LEARNING

Q-learning is one of the commonly used algorithms to solve the Markov Decision Process problem. The actions $a \in A$ are obtained for every state $s \in S$ based on an action-value function called

$$Q : S \times A \rightarrow R.$$

The Q-learning algorithm is based on the Bellman equation:

$$Q_{t+1}(s+a) \leftarrow Q_t(s+a) + \alpha[r + \gamma \; argmax \; Q_t(s', a') - Q_t(s, a)]$$

3.4 DEEP Q LEARNING

Constructing Q - table for complex problems like a self-driving car can be very difficult in the real world. In literature, researchers have used Neural Network to approximate the Q function. Neural networks are exceptionally good at coming up with good features for highly structured data. For this project we will use Deep Q Learning, i.e. we will combine Q-learning with Artificial neural network. We will encode the suitable states of the model as a vector and then pass it as input to the neural network. Outputs of the network are Q-values for each possible action. Then this loss error is back-propagated into the network, and the weights are updated according to how much they contributed to the error.

It is, however, important to point that the Q values updates happen in a different way than traditional Q-table approach. A loss error is calculated which is square of the temporal difference.

3.5 EXPERIENCE REPLAY

When we consider transitions from one state s_t to the next state s_{t+1}. The problem with this is that s_t is most of the time very correlated with s_{t+1}. Therefore, the network is not able to learn much. This could be improved if, instead of considering only this one previous transition, we consider last m transitions where m is a large number. This pack of the last m transitions is what is called the Experience Replay. Then from this Experience Replay, we take some random batches of transitions to make our updates

3.6 THE WHOLE DEEP Q LEARNING PROCESS

The whole steps of the Deep Q Learning process used in this project can be summarized as Initialization
For all couples of actions a and states s, the Q-values are initialized to 0:

$$\forall_a \in A, s \in S, Q_0(a, s) = 0$$

The Experience Replay is initialized to an empty list M.
We start in the initial state s_0. We play a random action and we reach the first state s_1.

At each time $t \geq 1$

1. We play the action a_t, where a_t is a random draw from the W_s distribution:

$$a_t \sim W_{s_t}(\cdot) = \frac{\exp(Q(s_t,.))^{\tau}}{\sum a', \exp(Q(s_t,a'))^{\tau}}\ , with\ \tau\ \geq 0$$

2. We get the reward $r_t = R(a_t, s_t)$
3. We get into the next state s_{t+1}, where $s_{t+1}$is a random draw from the $T(a_t, s_t, .)$ distribution:
$$s_{t+1} \sim T(a_t, s_t, .)$$
4. We append the transition (s_t, a_t, r_t, s_{t+1}) in M.
5. We take a random batch $B \subset M$ of transition. For each transition $(s_{t_B}, a_{t_B}, r_{t_B}, s_{t_B+1})$ of the random batch B:

   - We get the prediction:
   $$Q(s_{t_B}, a_{t_B})$$
   - We get the target:
   $$r_{t_B} + \gamma \max_a (Q(a, s_{t_B+1}))$$
   - We get the loss:
   $$LOSS = \frac{1}{2}(r_t + \gamma \max_a \left(Q\left(a, s_{t_B+1}\right)\right) - Q(a_t, s_t))^2$$
   $$= \frac{1}{2} TD_t(a_t, s_t)^2$$
   - We backpropagate this loss error and update the weights according to how much they contributed to the error.

4. IMPLEMENTATION

4.1 SIMULATION ENVIRONMENT

Model of the car and sensor is created as objects in Kivy and then imported in Python script, where the car moves in a rectangular map. The obstacle is also created using Python with an array that has as many cells as the graphic interface has pixels. Each cell has a one if there is an obstacle, 0 otherwise. The obstacle here is programmed in such a way that car speed will decrease as soon as it touches the obstacle. The values are updated per 1/60 seconds in the simulation environment i.e. 60fps which is generally for a game environment.

Load button on the screen can be used to load the last saved trained network. (A file is generated to save the training data and then updated each time the training is saved using save button on the screen).

4.2 DEEP Q NETWORK

The Neural network is created in Python using various libraries like NumPy and Torch. The torch came very handy as It provides a wide range of algorithms for deep machine learning. The network implemented here has only one hidden layer with 30 neurons.

The input layers have 5 neurons and the output layer have 3 neurons. The 5 input which are fed to the neural network.

Input 1: Signal from Front Sensor.
Input 2: Signal from Left Sensor.
Input 3: Signal from Right Sensor.
Input 4: Orientation of the car with respect to the goal.
Input 5: Negative of the Orientation of the car with respect to the goal.
These 5 input values describe the state of the environment.

There are 2 full connections in total in the network with the linear function used for connections, one between input and hidden layer, the other in between hidden layer and output layer. Then a forward function for activating the hidden neurons. The rectifier function was used as an activation function.

A number of actions: 3 (Move Left, Move Right, Move straight).

Rewards:

The car gets a reward of:

- -1 when it is in contact with the obstacle
- 0.1 when the car is moving closer to the goal.
- -0.2 when it moves away from the goal.
- -1 when it’s near the either of the four edges of the map.

Discount Factor: 0.9

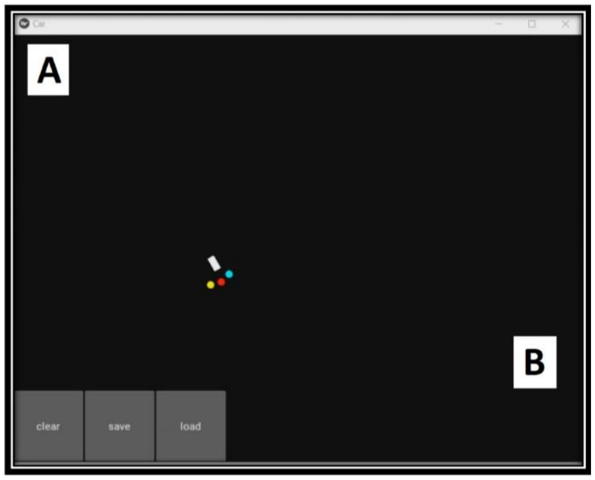
5. RESULT

The source code of this project is available on GitHub.

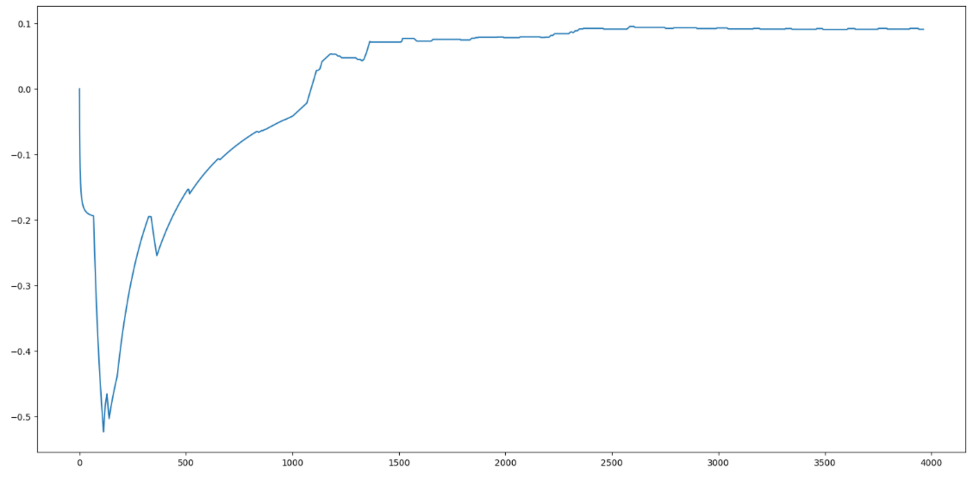
Three types of task were used to simulate the car where it performed very well

5.1. TASK 1

In task 1, the car is assigned the task of moving from point A to B in the map. There are no obstacles in the map for this task.



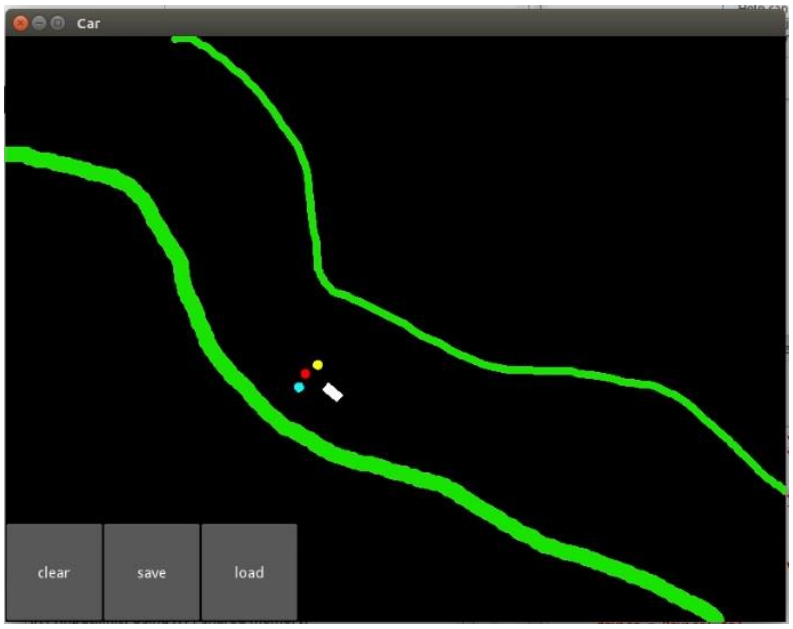
In task 1, the car achieved the maximum reward of 0.1 in 4000 simulation run time.



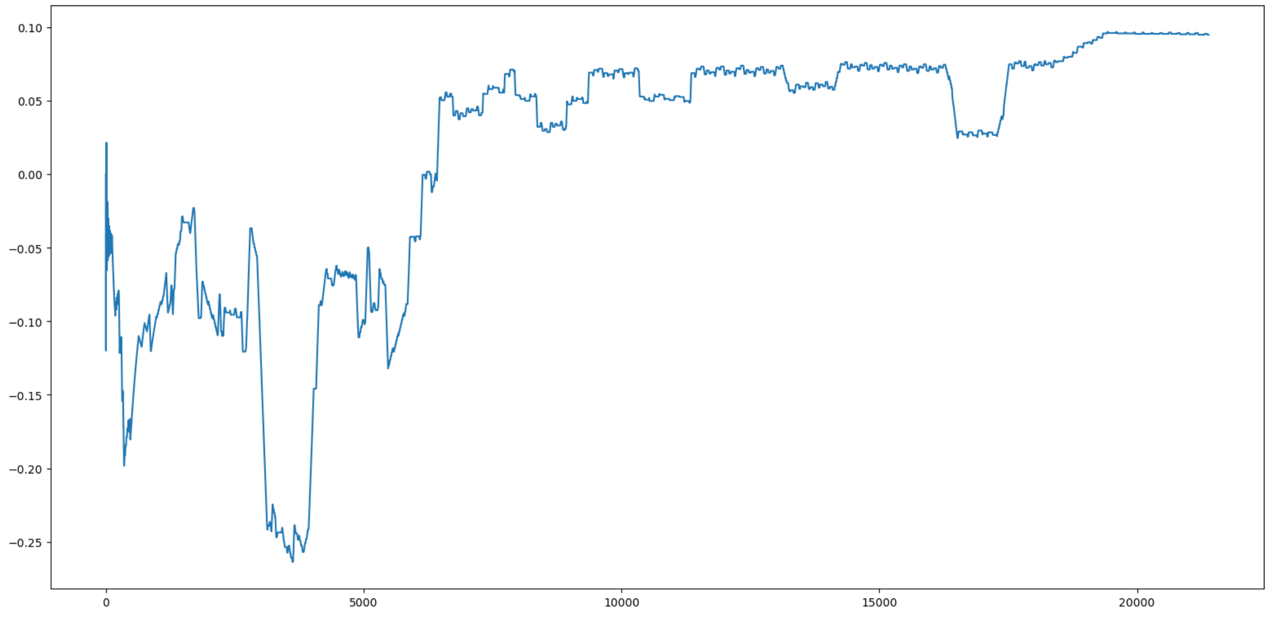
Map and Reward vs Time Plot for task 1

5.2. TASK 2

Task 2 was to move the car from point A to B in the map while moving in a confined road.



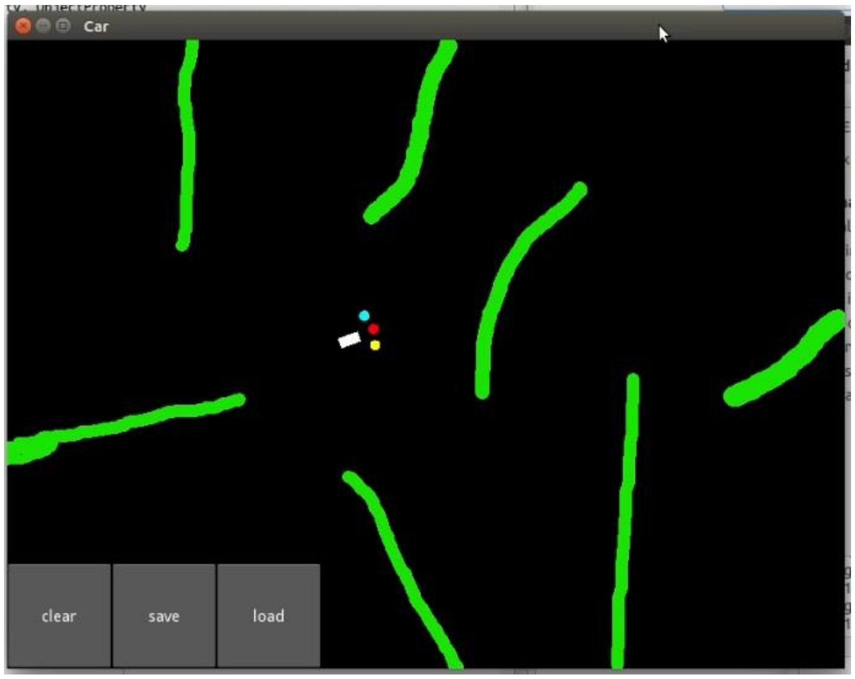
In task 2, the car achieved the maximum reward of 0.1 in 20,000 simulation run time.



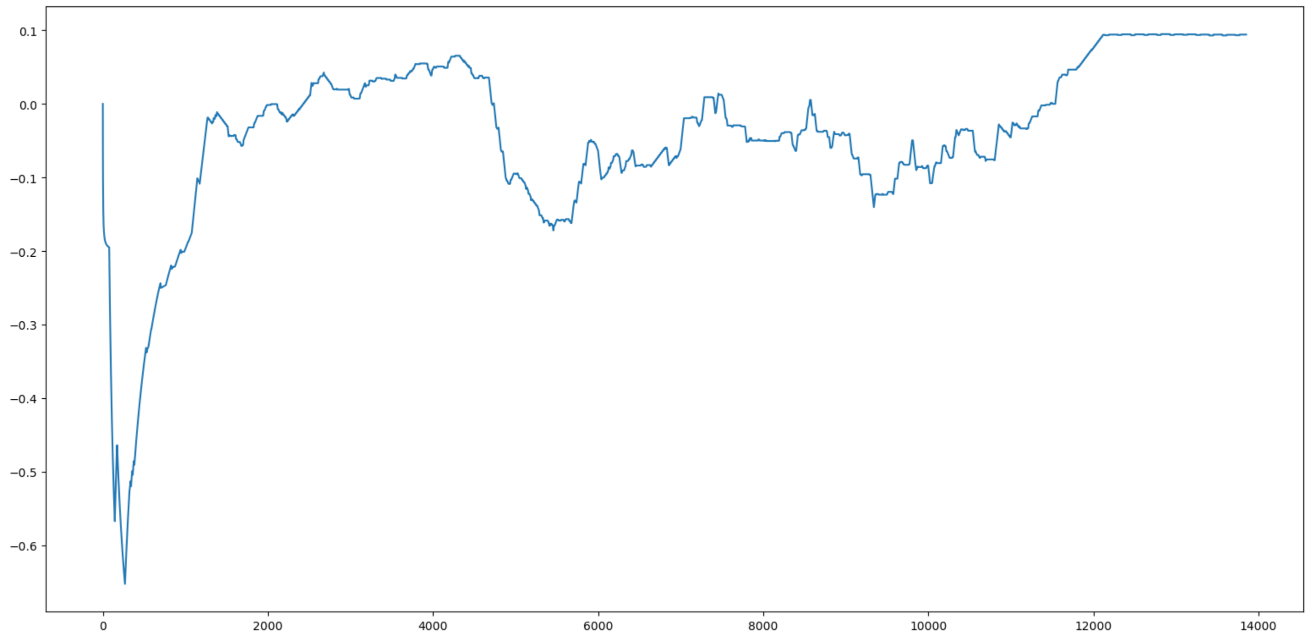
Map and Reward vs Time Plot for task 2

5.3 TASK 3

Task 3 was to move the car from point A to B in the map while avoiding the collision with obstacles in between.



In task 3, the car achieved the maximum reward of 0.1 in 13,000 simulation run time.



Map and Reward vs Time Plot for task 3

6. ANALYSIS

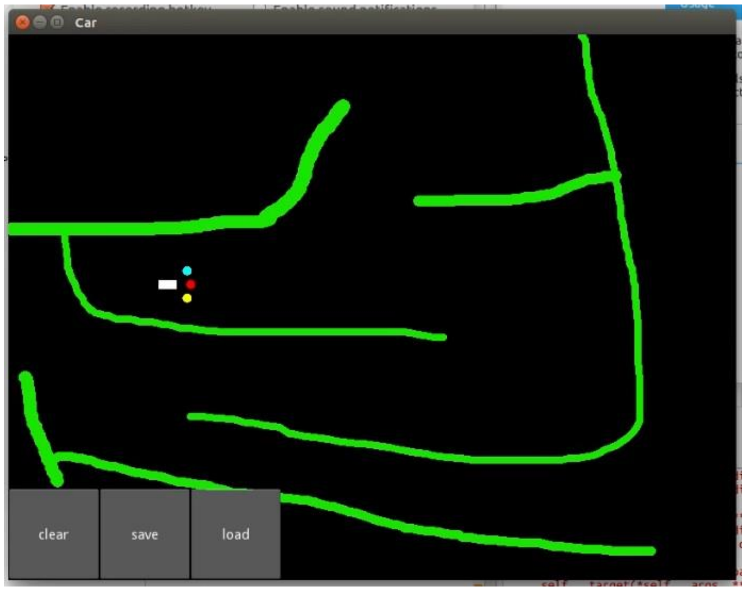
The car seems to perform very well in the current neural network architecture and rewards parameter. But, coming up at the optimum architecture is a not an easy task. Selection of 30 neurons in the hidden layer was done after some trial and error by varying the number of neurons. Selection of reward is also critical in deciding the learning convergence. A small positive reward of 0.1 when the car moves towards the goal, lets the car to achieve the task and not collide with an obstacle. When this reward was set to 1, the car did not achieve the maximum reward after so many iterations also.

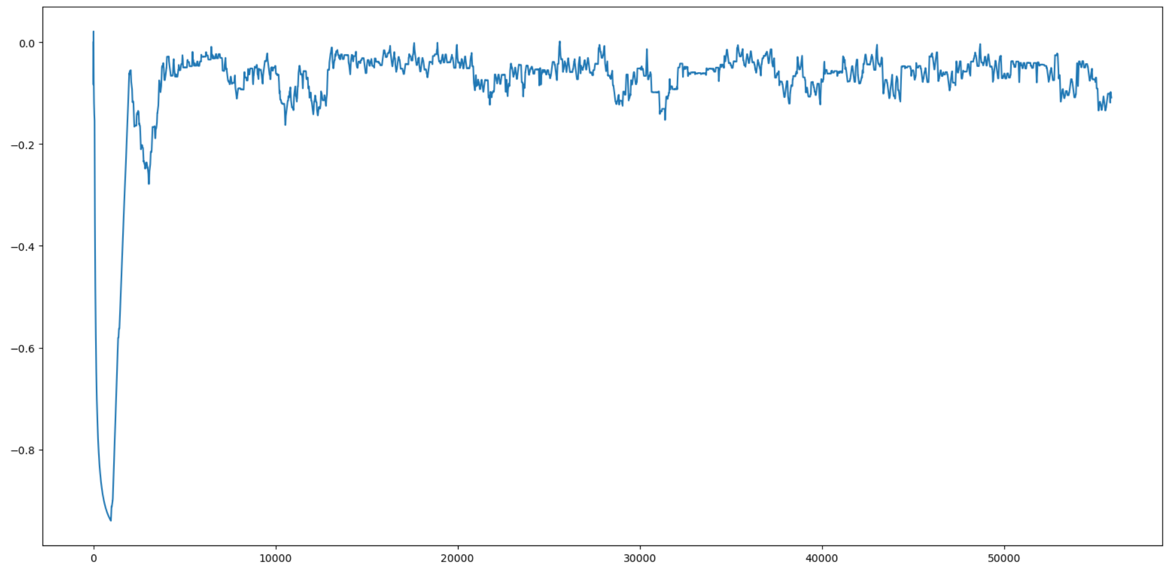
7. CONCLUSION

In this project, Deep Q learning was used to develop a model of the self-driving car. With the given simplicity of neural network of only one hidden layer, the car performed significantly well and learn very quickly. The car achieved the maximum reward available in all three tasks. Use of experience replay was significant as it helped the car to learn quickly especially where there was a requirement of a sharp turn

8. FUTURE WORK

The car performed well in all the three desired tasks, but when the map was made more complex, it was not able to achieve the maximum reward even after more than 50,000 episodes.





Map and Reward vs Time Plot for the complex task

It will be a great challenge for the readers to achieve this and try various changes to achieve the task.
Possible solutions can be:
Addition of more Hidden Layers.
Addition of states (Maybe time constraint using a timer)
Change in rewards.

9. REFERENCES

[1] https://www.intel.ai/demystifying-deep-reinforcement-learning/

[2] https://kivy.org/doc/stable/tutorials/pong.html

[3] Reinforcement Learning for Autonomous Vehicles by Jeffrey Roderick Norman Forbes https://www.cs.duke.edu/forbes/papers/thesis.pdf

[4] Pomerleau, Dean A. Alvinn: An autonomous land vehicle in a neural network. No. AIP-77. CARNEGIE-MELLON UNIV PITTSBURGH PA ARTIFICIAL INTELLIGENCE AND PSYCHOLOGY PROJECT, 1989.

[5] Deep Reinforcement Learning for Simulated Autonomous Vehicle Control April Yu, Raphael Palefsky-Smith, Rishi Bedi Stanford University

[6] My GitHub Repository - https://github.com/gkkhut/Deep-Reinforcement-Learning-for-Simulated-Self-Driving-Car