Design and Simulation of an Artificial intelligence (AI) Brain for a 2D Vehicle Navigation System

Raja MUNUSAMY*[,1,a], Akash MUKHERJEE[1,b], Kokila VASUDEVAN[1,c], Balaji VENKATESWARAN[1,d]

*Corresponding author
[1]Aeronautical Department, Hindustan Institute of Technology and Science,
Rajiv Gandhi Salai (OMR),
Padur, Kelambakam, Chennai -603103, Tamil Nadu, India,
rajam@hindustanuniv.ac.in*, Akash@gmail.com, kokilaspv@gmail.com,
balajivenkateswaran.v@gmail.com

DOI: 10.13111/2066-8201.2022.14.2.5

Received: 28 August 2021/ Accepted: 19 April 2022/ Published: June 2022

Abstract: *The aim of this research paper is to develop an artificial intelligence (AI) brain for a 2D vehicle to help it navigate in a certain environment. The goal of the 2D car is to do round trips between two defined checkpoints while avoiding obstacles that are created by the user. The two defined checkpoints are the top left corner of the screen and the bottom right corner of the screen. This project makes use of PyTorch API to design the brain of the car and of Kivy API to design the environment of the car. Spyder, a cross-platform integrated development environment is used to visualize the project.*

Key Words: *Artificial Intelligence, Deep Reinforcement Learning, Intelligent Vehicle, Deep Q-Learning*

1. INTRODUCTION

Artificial Intelligence (AI) has become a very important part of our daily lives as we head towards the end of this decade. Artificial Intelligence existed in the 60's, 70's and got picked up in the 80's and lot of research was done in this field and everyone thought it was this new thing that would change the world completely but it eventually died down. The reason for the slowdown was due to technology constraints at that time as computing power was not up to the required standards to facilitate Neural Networks [1].

In order for Neural Networks to work properly, they need huge amounts of data and lots and lots of computing power. This is the reason for revived interest in this field as now we have two new emerging fields, namely- Big Data and Deep learning, where companies have lots of data on their customers and deep learning is a part of Artificial Intelligence that requires these data and lots of computing power to train it to make predictions [2].

[a] Associate Professor-Dr
[b] Faculty
[c] Faculty
[d] Faculty

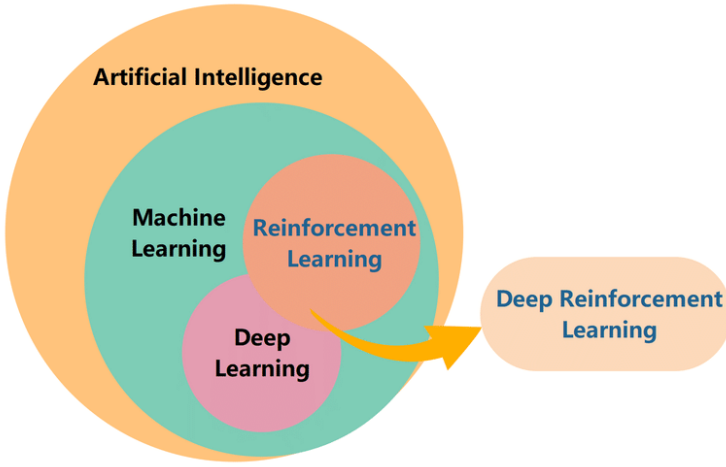

Fig. 1 The relationship between AI, ML, RL, DL and DRL

Artificial Intelligence is a very broad term. In this research work, we dive into the very cutting-edge algorithm of artificial intelligence that is being used currently all over the world of AI, namely- Deep Reinforcement Learning. We use an algorithm that is called the Deep Q-Learning Algorithm. It is derived from Q-learning that is a reinforcement learning algorithm and the concept of ANN that is in deep learning.

2. LITERATURE REVIEW

[1] The Theory of dynamic programming, Richard Bellman: This paper gives a deep understanding of the Bellman Equation, which forms a base of our project. This is a very old paper. It lays the foundation of some of the terminologies that we use in our paper and it discusses the fundamental approaches and mathematical formulation of the equation, which we now call the Bellman equation.

This equation is discussed in detail further in our paper. [2] A survey of applications of Markov Decision Processes, D. J. White: This paper gives a list of scenarios where, MDP may be applicable and the results to help better understand the use case of MDP and designing of problem in accordance.

[3] Markov Decision Processes: Concepts and Algorithms, Martijn van Otterlo: This paper gives the required intuition and concepts behind Markov Decision Process and learning in sequential decision-making problems.

Our project uses the very concept of sequential decision and sequential neural networks to arrive at the decisions.

[4] Learning to predict by the methods of temporal difference, Richard S. Sutton: This paper gives a deep understanding of temporal difference and advantages in real-world prediction problems like less memory usage, etc.

[5] Tom Schaul, John Quan, Ioannis Antonoglou and David Silver: This 2016 paper by Google DeepMind team is a very important paper on Experience Replay which we use in our project to learn from previous experiences without memorizing them.

This paper helps to develop a deep understanding of the concept to avoid common pitfalls such as memorizing the path.

3. METHODOLOGY

Simple Q-Learning can be used for very simplistic environments but it is not powerful enough for things like making an AI play games, teach self-driving (our case), movement of robots (robotics). Therefore, we capitalize on the power of artificial neural networks and in-depth Q-learning. In simple Q-Learning, the agent has a before value (the empirical evidence) and it calculates temporal difference with (after value – before value) and minimizes the TD to achieve its goal [6]. This essence completely changes in Deep Q-Learning as there is no after in play. In Deep Learning, the input states (that describe the agent's position, etc., in the environment) are passed through the neural network, a weighted sum is calculated, activation function is applied and Q-values are predicted by the neural network for that specific state. Now, we have the empirical evidence from previous iterations in memory, of that specific state. We compare the target Q-values (empirical evidence) to the predicted Q-values to calculate a loss function, i.e., the summation of squared difference of the predicted and target Q-values. Ideally, we wanted the TD to be zero, but that can never be, therefore we aim for as close to zero as possible. In addition, we want the loss to be as close to zero as possible. Here, the difference arises and we harness the power of ANN [7]. We are going to take the loss and back propagate it through the network and through stochastic gradient descent update the weights of the synapses in the network so that next time we go through the environment; the weights are better descriptive of the environment. This process iterates repeatedly trying to make the loss minimum and that is how the ANN learns through Deep Q-Learning. This covers the learning part but this does not tell us how the ANN selects an action.

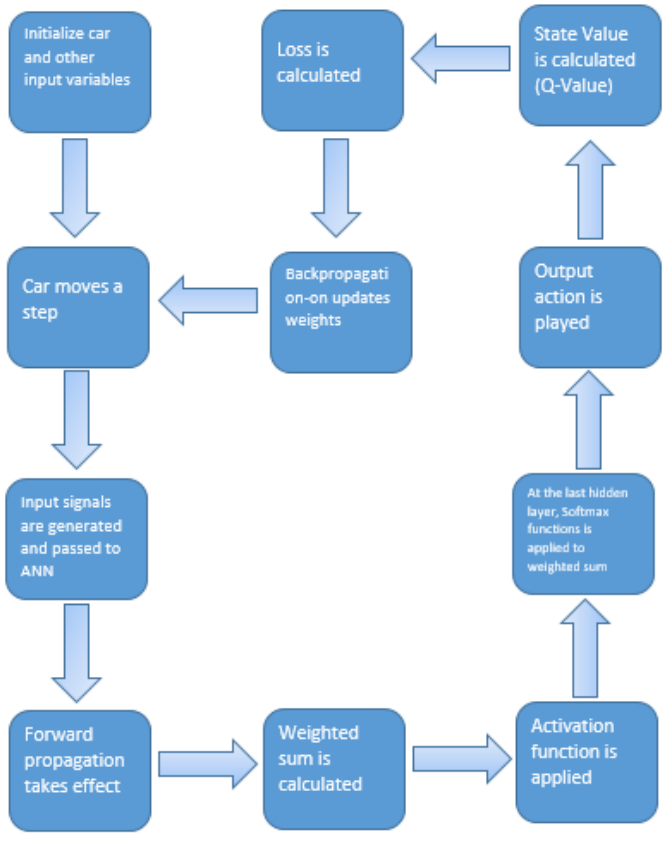

Fig. 2 ANN learns through Deep Q-Learning

The action selection is done by passing all the predicted Q-values through a softmax function. This process happens at every state of the environment and once the environment goal is reached, the agent is said to complete one epoch. After each epoch, the whole process is repeated.

4. ALGORITHMS

4.1 Reinforcement Learning

An agent (the AI) performs certain actions in a given environment. This leads to change in state (of the agent with respect to the environment), i.e., the parameters that define the state change because of the action the Agent takes and it gets rewards based on the action. By repeating the process, the agent learns about the environment, more specifically, which actions lead to favorable states and good rewards or unfavorable states and bad rewards [8].
Bellman Equation:

$$V(s) = \max_a(R(s,a) + \gamma V(s')) \quad (1)$$

where, V(s) → value of current state,
R(s,a) → reward of taking a certain action in the current state,
V(s') → Value of the following state,
γ → discounting factor.

Deterministic Vs Non-Deterministic Search – A deterministic search in our case means when the agent wants to move in a certain state by taking a specific action, it moves to the said state with a 100% chance. Whereas in a non-deterministic search the agent doesn't have a 100% chance that its action leads to the said state but a little less. Now, this is used to introduce some randomness to the environment which makes the environment closer to a real-world scenario.

4.2 Markov Process & Markov Decision Process

A stochastic process has the Markov property if the conditional probability distribution of the future states of the process (conditional on both past and present states) depends only on the present state and not on the sequence of events that preceded it. A process with this property is called a Markov Process [9].

Markov Decision Processes provide a mathematical framework for modelling decision making in situations where outcomes are partly random and partly under the control of a decision maker (agent in our case).

Therefore, our agent uses MDP to guide itself through an environment, which has properties of a Markov Process. This slightly changes the Bellman equation shown above, as now the agent does not know which/what state it can reach and only knows the probability of the states [9].

$$V(s) = \max_a(R(s,a) + \gamma . \sum_{s'} P(s, a, s')V(s')) \quad (2)$$

4.2.1 Policy Vs Plan

A plan is developed when everything is under the control of the agent. The original Bellman equation helps to develop a plan. But as soon as stochasticity is introduced in an environment, a plan cannot be executed as some factors are now out of the agent's control due to the stochasticity of the environment. This is where a policy is developed by the agent. This policy sometimes includes moves or actions that a human does not think of easily. A very famous

example may be the AlphaGo AI that defeated the GO world champion in Korea where it came up with moves that the humans did not know or hadn't used in many years of playing the game.

4.2.2 Living Penalty

It incentivizes the agent to complete the task at hand more quickly. The amount of living penalty imposed on the agent in-turn influences the policy developed by the agent to complete the task. For example, if there is no or very low living penalty, the agent may prefer to take the longer policy in order to complete the task. Now, if the living penalty imposed is very high, the agent might change its policy to take a bad action at a certain state as reaching the goal (which may be far) would result in incurring and accumulation of high living penalty, thus the agent would decide not to reach the goal, instead take a bad action to prevent getting worse living penalty than the bad reward (which the agent might think is now better as compared to the living penalty). Obviously, both of the situations mentioned above are less desirable, and thus a modest living penalty is applied to the environment to help the agent frame a policy that might take risks to get a reward, rather than apply a more secure policy to achieve the goal.

4.2.3 Q-Learning Algorithm

It converts the equation from being in terms of state values to being in terms of quality of actions taken from a certain state. Q-Learning is a way to quantify the quality of action in a current state. This helps quantifying all the actions that are available in that state to help pick the best action [10].

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} (P(s, a, s') \max_{a'} Q(s', a')) \quad (3)$$

Thus, the agent, instead of looking at possible states it may end up in, compares the possible actions and decides the best action it may take.

4.2.4 Temporal Difference

We have to look at temporal difference in a step by step manner. The Formula for Q-Learning is:

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} (P(s, a, s') \max_{a'} Q(s', a')) \quad (4)$$

for the sake of simplicity, we give up the probabilities of action from the formula but the essence would always include the action probabilities. Therefore, we re-write the above equation as –

$$Q(s,a) = R(s,a) + \gamma \max_{a'} Q(s', a') \quad (5)$$

Therefore, the temporal difference is defined as follows:-

$$TD(a,s) = R(s,a) + \gamma \max_{a'} Q(s', a') - Q_{t-1}(s,a) \quad (6)$$

Ideally the RHS of the equation should result in zero but the Q(s,a) in RHS is a result of previous iterations, or might I say empirical evidence whereas the rest of the term is calculated immediately after a certain action is selected, which is a part of the current iteration [11]. The reason for both not cancelling each other out is because of the environmental stochasticity. Both entities represent the same thing, i.e., Q value of a state, the difference consisting in the time of calculation. Here, discarding the old Q-value because it is old and different from the new may not be smart as the old Q-value may have been a collective of the dominant probability and the new Q-value may be due to randomness. In this case, we would throw

away the Q-value that is responsible for the bulk of the time and keep the one that is due to randomness, obviously not smart. Therefore, we use temporal difference to update Q-value we use temporal difference as follows [11]:

$$Q_t(s,a) = Q_{t-1}(s,a) + \alpha TD_t(a,s) \quad (7)$$

Here, $\alpha \rightarrow$ Learning rate.

4.3 Perceptron/ Neuron

The neuron is a unit in the human brain, which is connected to thousands of other neurons, and billions of them are interconnected to form the intricate design of the brain. A perceptron is shown below:

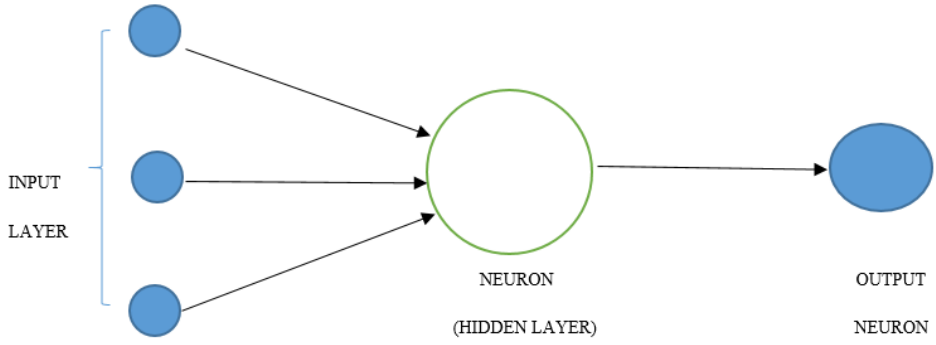

Fig. 3 Representation of neuron (Perceptron)

How a neuron works is by taking input signals from sensors of the body(sight, touch, sound, smell and taste) and when a sufficient input is received, the neuron 'activates' and fires its synapses(the connection between neurons) to its adjacent neurons to produce an output that becomes the input of the next neuron and so on and so forth.

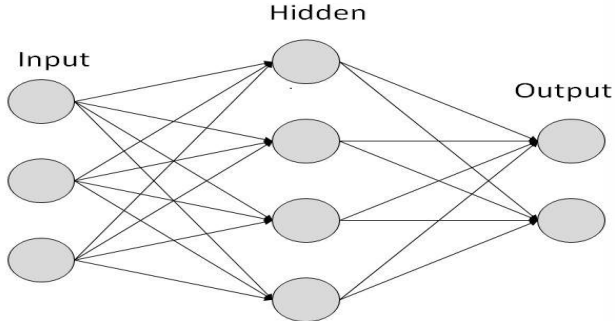

Fig. 4 Artificial neural network

4.3.1 Artificial Neural Network (ANN)

The perceptron is a digital representation of neuron that is used to build the artificial neural network. Each input signal carries with it some weights that are initialized randomly but updated throughout the learning process. The hidden neuron takes the inputs multiplied with its respective weights, sums them all up and applies an activation function to the summed multiplication. This mimics the synapse.

Learning Process – After every step mentioned above is completed, the supply of input values is then calculated as a weighted sum, then applying the activation function to produce the

output, the predicted output value of the neuron is compared to an actual expected value using a loss function. There are many loss functions but most common is subtracting the actual value from the predicted value and squaring it to spit the value of the cost function [11]. Now the goal is to minimize the cost function. This is done by adjusting the weights of the neurons according to the cost function to minimize it and as soon as the cost function reaches a minimum after several iterations with updating of weights, the training is stopped with the final weights being the optimal values. The process of updating the weights according to the cost function is called back propagation.

4.3.2 Stochastic Gradient Descent

It is the iterative process used to update the weights. There is a problem of our AI getting stuck in local-minima where it thinks the minima it has achieved is the best but it is not. This happens when the cost function is not convex, which is the maximum number of cases, there stochastic gradient descent is used as it doesn't require the cost function to be convex and with decent step-size we can find the global-minima.
Experience Replay is an important aspect of the learning part of the ANN. The environment may have certain experiences that occur less and certain that occur more and we have seen how important it is for the ANN to have repeated experiences to learn properly. Otherwise it may lead to the agent performing certain actions with high accuracy and certain actions with low accuracy (because of less experience or less opportunity to learn because of design of environment). Experience Replay works by saving n-number of experiences (s_t, a_t, r_t, s_{t+1}) in a batch and selecting randomly from those experiences and feeding it to the ANN to learn from them so as to break the pattern of dependence and correlation of sequential experiences. Also, the network can learn faster because it has the limited experience saved and doesn't need to go through as many epochs [12].

4.3.3 Action Selection Policy (the softmax)

On the face of it, action selection seems like a rather easy task, just select the action with the highest Q-value, but this does not work out well. The reason for this is to assume that the agent has 4 Q-values to choose from. Moreover, that stage of training it chooses a certain Q-value that is max at that stage but it is fully possible that another Q-value out of the remaining 3 values gives even better results than the selected one. This is an instance of local-maxima. Now, the problem is if it is stuck in local-maxima, the agent may get a false sense of the Q-value being the best action simply because it does not explore the other actions. It may be possible for the agent to take several steps to see that there is a better action at that particular stage because of not exploring different options and further making its sense of action, being good, stronger [11], [12]. This causes bias in the network, which should be avoided. There exists a principle of Exploration VS Exploitation. The essence of the principle is 'Exploitation' of a good Q-value is acceptable, but this should not stop the network to not 'Explore' further for a better Q-value whose 'Exploitation' may give the network better results. The softmax function helps in this respect. The formula for the function is as follows:

$$f_j(z) = e^{z(j)}/\sum_k . e^{z(k)} \quad (8)$$

The softmax function does is squeeze the output values of the network in between 0 and 1 such that the values add Upto 1. This is a very important property that we will make use of. The values after softmax transformation can now be interpreted as probabilities. Now, let's jump back to our example of 4 Q-values. Let us assume the values for action 1 (Q(1,s))=0.05, action 2 (Q(2,s))=0.90, action 3(Q(3,s))=0.02 and action 4 (Q(4,s))=0.03 after passing through

softmax function. Now we can interpret these values as the probability of action 1 being the best action is 0.05, action 2 is 0.90, action 3 is 0.02 and action 4 is 0.03. To preserve Exploration in our model, we use these exact probabilities as the number of times certain action is played. To elaborate further we can say that the network will choose to play action 1 5% of the time, action 2 90% of the time, action 3 2% of the time and action 4 3% of the time [12].

4.3.4 Kivy cross-platform design

Kivy is a cross-platform game development environment that is used to make the environment, the car, the sensors of the car and the obstacles. The whole game, that our AI plays, design is divided into two parts. In one part we design the car, basically a solid rectangle which is given motion. Then we design the sensor representation, i.e., three circular balls that represent the three sensors of the car. In the second part, we design a paint application, which then enables us to draw obstacles using the mouse, which we call sand. The working of sensors is then established like so: an area of 20x20 pixels is takes around each sensor, which gives us 400 pixels. The background of the environment is kept black, or 0 valued pixels. In paint app we specify the colored pixels to be of value 1. Then sensor is coded to sum all non-zero, i.e. all one's, pixel values and divide it by 400. This division by total no. of pixels in sensor area gives us the sand density around that sensor.

Rewards are an essential part of the Neural Network working as they provide a sense of direction, do's and don'ts, to help the Network reach its goal again and again. Rewards are hyper parameters, meaning their optimum values are decided by trial and error method. For, Neural Network, there are a set of positive and negative rewards that have been set as follows the positive rewards for the AI are for reaching checkpoint starting from the checkpoint. When distance between car and goal reduces [13]. The negative rewards are differential time reward for staying alive (starting from checkpoint time – reaching checkpoint time)/ 100, which prevents the AI from forever exploring, hints to finding the shortest path, and is subtracted from total reward each time. For reaching Edge of map (left, right, top and bottom) and for going into sand, the AI gets negative reward. The maximum negative reward is -3 and max positive reward is +2.

5. RESULTS

Level 1 (a):

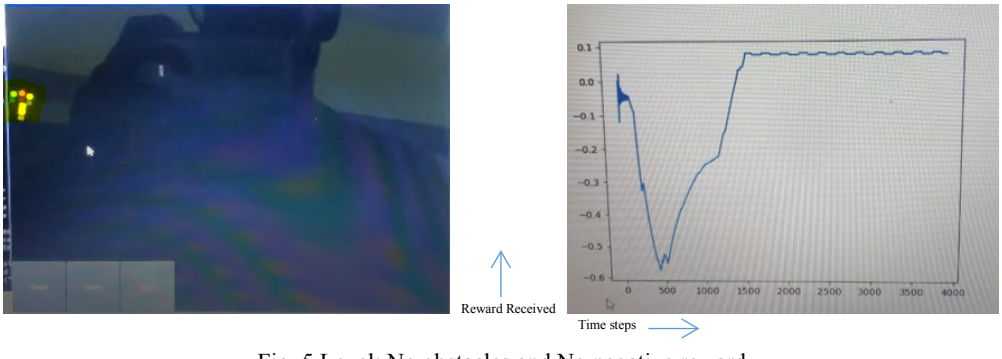

Fig. 5 Level: No obstacles and No negative reward

The x-axis of graph is time steps by the AI; The y-axis is the reward received by the AI.
Level 1(a) – this level has no obstacles and no negative reward in the form of differential time, for time spent in between checkpoints.

The early stage of the graph clearly depicts a series of bad actions performed by the AI and it quickly learns what is required of it, thus showing a spike in rewards through time. As the reward calculated is cumulative in nature, it steadies out near +0.05.

Level 1 (b):

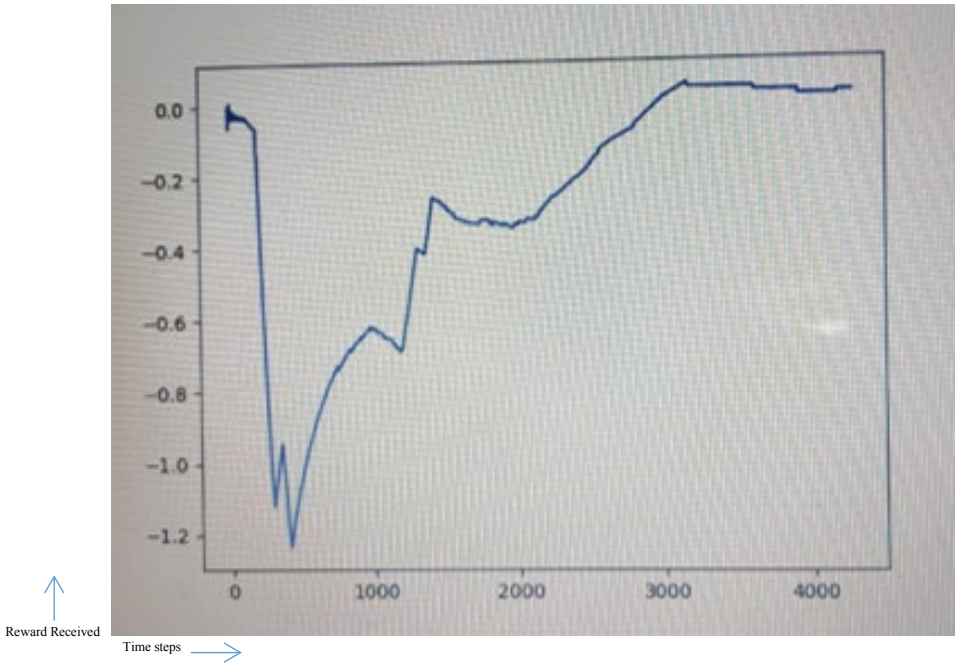

Fig. 6 Rewards and some hyper parameters are tuned

The map for level 1(b) is same but rewards and some hyper parameters are tuned. Main difference is the addition of negative differential time reward for exploring.
We can see every downward dip of the curve as mistakes and how fast the AI rectifies them with the upward climb.

Level 2:

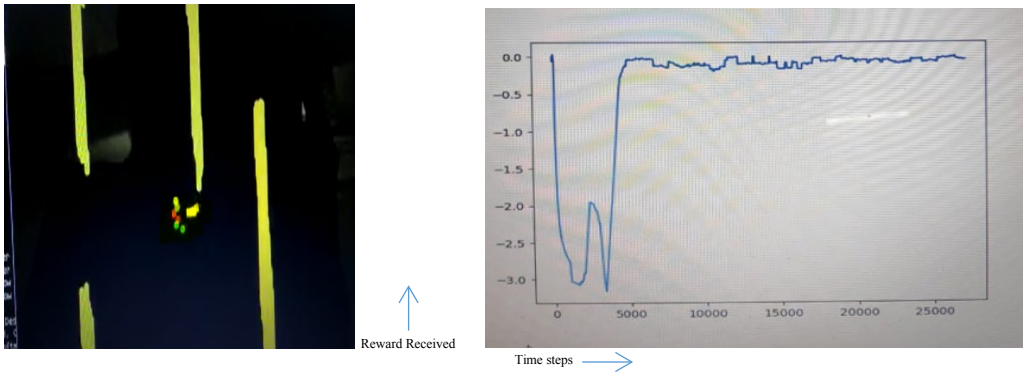

Fig. 7 Obstacles between the two checkpoints

This level contains some obstacles between the two checkpoints of the map, namely the top left corner and bottom right corner.

Level 3:

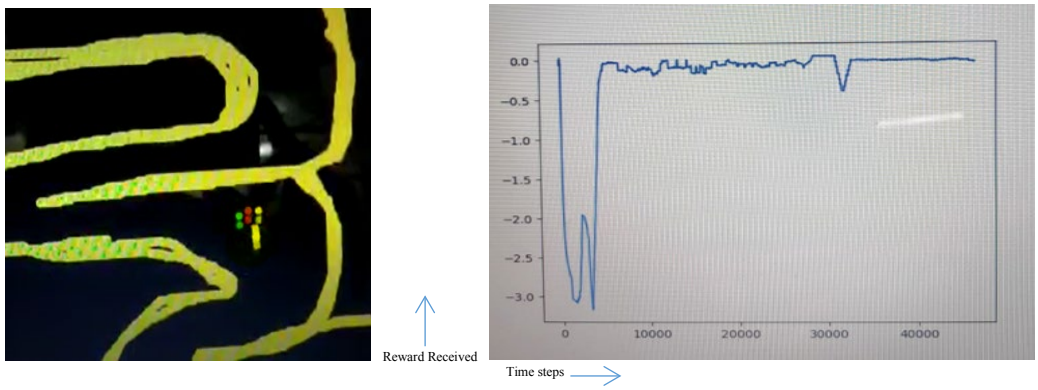

Fig. 8 Zero slope over exploitation according to the softmax probabilities

The car gets a slight positive reward every time it orients itself towards the goal, i.e., every time the distance between and goal reduces. Also, in graph for level 3 we can see a sudden dip after the stage of zero slope. This is actually a mistake due to the AI choosing exploration over exploitation according to the SoftMax probabilities. Therefore, it is our job to stop training at the time we achieve desirable results to keep AI from exploring unnecessarily.

From each of the curves we see that there are 3 general trends in the graph. The negative slope regions, the positive slope regions and a region of zero slope. The negative slopes show mistakes made by the car, which result in negative rewards, and hence the curve dip in those regions. The mistakes may be going onto some sand, reaching the edges of the map or exploring excessively. The positive slopes show a series of actions resulting in positive rewards, or in other words performing actions that lead to positive rewards like reaching a checkpoint or reduction in distance between car and goal.

Level 4:

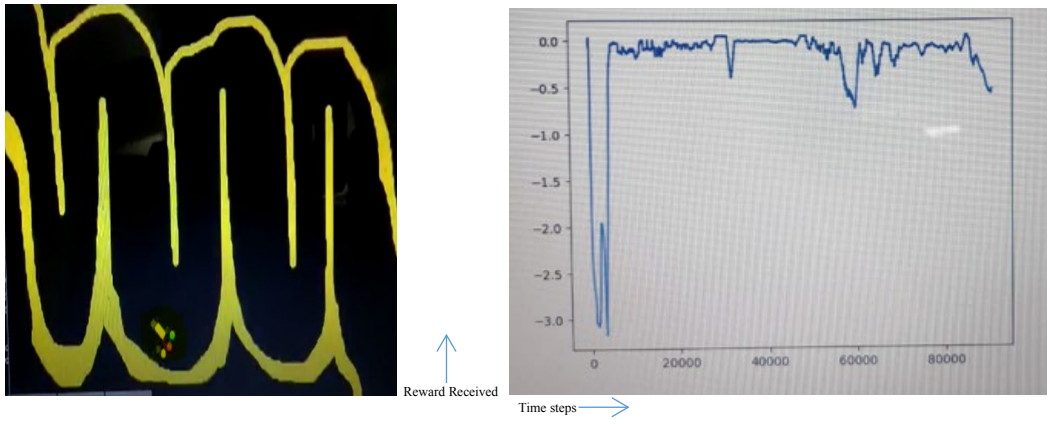

Fig. 9 Negative reward proportional to the time spent between two checkpoints

The zero slope indicates the constant reward. This can be deduced as the point at which the AI performs the best set of actions that it believes leads to minimum loss in the loss function. But we see in levels 4 and 5 that the zero slope keeps fluctuating. This can be explained by the obstacle design. As the car gets a continuous negative reward proportional to the time spent between two checkpoints, the longer the maps – the more negative the rewards keep getting.

Level 5:

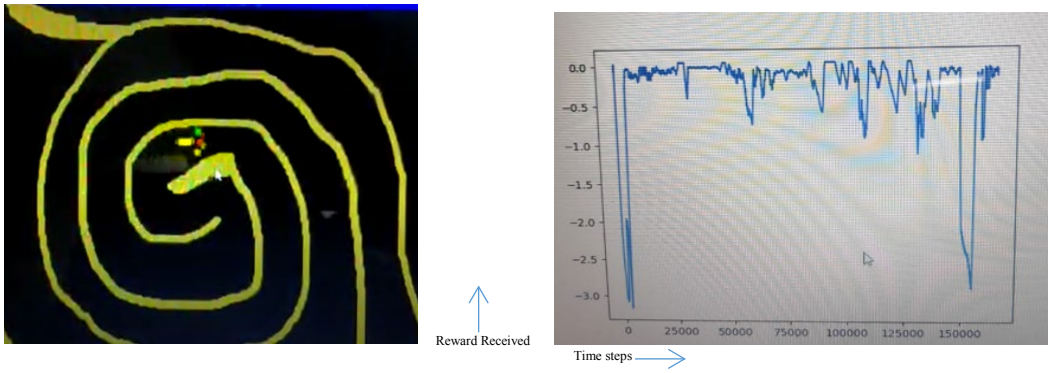

Fig. 10 More negative the rewards keep getting (between two checkpoints)

From each of these curves we can see that even after learning what to do (averaging out) the AI is still prone to mistakes. This is partly because it never stops exploring and due to softback probability, it can sometimes randomly select actions. Also, it is up to us to stop the training when we think the AI has maxed out. The averaging out is in other levels is not as smooth as level 1(a) or (b) because of the length of the maps. The more time AI spends between the two checkpoints, it keeps getting a negative reward.

6. CONCLUSIONS

In this study, two key branches, namely, Deep Learning and Reinforcement Learning of ML and AI were analyzed and implemented by focusing on the development of self-driving vehicle and the previous research on combining the two streams via Deep Q-Learning. Deep Learning is learning from a training set and applying to a new set, whereas, Reinforcement Learning is continuous learning by adjusting actions based on continuous feedback. Q-Learning is a Reinforcement Learning algorithm. In our project, we successfully implement Deep Q-Learning algorithm which combines the best of both. We use the base of reinforcement learning (give rewards as feedback to actions performed) and introduce the experience replay to learn from previous experiences (analogous to training set) to achieve a self-driving vehicle.

REFERENCES

[1] A. Juliani, Simple *Reinforcement Learning with Tensor flow Part 0: Q-Learning with Tables and Neural Networks*, Editor of Beyond Intelligence, (https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-and-neural-networks-d195264329d0), 2017.

[2] R. S. Sutton, A. G. Barton, *Reinforcement Learning I: Introduction*, Citeseer, (http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.32.7692), 1998.

[3] R. Bellman, *The theory of dynamic programming*, P 550, The Rand Cooperation, California, (https://www.rand.org/content/dam/rand/pubs/papers/2008/P550.pdf), 1954.

[4] D. J. White, A Survey of Application Markov Decision Process, *The journal of operational research society*, Vol **44**, No 11, PP 1073-1096, http://www.cs.uml.edu/ecg/uploads/AIfall14/MDPApplications3.pdf, 1993.

[5] Martijn van Otterlo, *Markov Decision Processes: Concepts and Algorithms*, SIKS course on (https://pdfs.semanticscholar.org/968b/ab782e52faf0f7957ca0f38b9e9078454afe.pdf) Learning and Reasoning – May 2009.

[6] R. S. Sutton, Learning to predict by the methods of temporal differences, *Mach Learn* 3, 9–44 (1988). https://doi.org/10.1007/BF00115009 (https://link.springer.com/article/10.1007/BF00115009)

[7] R. S. Sutton & A. G. Barto, A temporal-difference model of classical conditioning, *Proceedings of the Ninth Annual Conference of the Cognitive Science Society* (pp. 355–378), Seattle, WA: Lawrence Erlbaum, 1987.

[8] R. J. Williams, *Reinforcement learning in connectionist networks: A mathematical analysis* (*Technical Report No. 8605*), La Jolla: University of California, San Diego, Institute for Cognitive Science, 1986.
[9] A. Juliani, *Simple Reinforcement Learning with Tensor flow Part 4: Deep Q-Networks and Beyond*, (https://medium.com/@awjuliani/simple-reinforcement-learning-with-tensorflow-part-4-deep-q-networks-and-beyond-8438a3e2b8df), Editor of Beyond Intelligence, 2016.
[10] T. Schaul, J. Quan, *Prioritized Experience Replay*, ICLR conference 2016, (https://arxiv.org/pdf/1511.05952.pdf) (http://neuralnetworksanddeeplearning.com/chap2.html)
[11] M. Tokic, *Adaptive ε-greedy Exploration in Reinforcement Learning Based on Value Differences*, Institute of Neural Information Processing, University of Ulm, 89069 Ulm, Germany, http://tokic.com/www/tokicm/publikationen/papers/AdaptiveEpsilonGreedyExploration.pdf
[12] X. Glorot, Deep Sparse Rectifier Neural Networks, *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, PP 315 -323, 2011.
http://jmlr.org/proceedings/papers/v15/glorot11a/glorot11a.pdf
https://iamtrask.github.io/2015/07/27/python-network-part2/
[13] R. DiPietro, *A Friendly Introduction to Cross-Entropy Loss*, Version 0.1 – May 2, 2016,
https://rdipietro.github.io/friendly-intro-to-cross-entropy-loss/
(http://peterroelants.github.io/posts/neural_network_implementation_intermezzo02/)