Survey on Simulation of Self-Driving Cars using Supervised and Reinforcement Learning Models

Athul Dev[1], Roshni K S[2]

[1] Student, Department of Computer Science & Engineering, Nitte Meenakshi Institute of Technology, India
[2] Student, Dept. of Electrical & Electronics Engineering, Sir. M Vishveshvaraya Institute of Technology, India

---***---

Abstract - *A self-driving system is the heart, soul and future of the automobile industry and there is a lot of techniques and technology involved in the accomplishment of a self-driving car. Since the system has to be deployed in the real world and which is supposed to take multiple actions and sudden reactions in real time, the system should be tuned to a very high degree of precision. In this paper we share our observations with respect to tuning and tweaking most of the hyper parameters and the architecture in a generated environment for both supervised and reinforcement learning techniques. And how different a supervised learning model is to a reinforcement learning model with respect to the self-driving car. We use the concepts of Deep Learning, Deep Neural Networks, Convolutional Networks and Deep Q learning in order to configure the brain of our car to perform certain actions.*

Key Words: **Machine Learning, Supervised Learning, Reinforcement Learning, Q Learning, Convolutional Neural Networks, Deep Learning, Markov Decision Process.**

1.INTRODUCTION

From the perspective of a self-driving car, we have mainly considered the motion, the movement and the overall physics of the car. In the supervised model we have made the prediction of the car's position, its steering angle and the speed of the car in the environment based on how we drove the car in that particular environment, so on the basis of certain frames of images encountered, the required steering angle and speed in fed back to the car as it's input. That is image is the input and the steering angle with the speed, throttle and brake is the output. So in this case we are not considering any sorts of external obstacles like another vehicle, a barricade and so on. Whereas in the reinforcement model we have considered the obstacles by giving it a negative reward when it encounters and obstacles or barriers, due to which it tends to avoid these obstacles. We have tuned various hyper parameters like the learning rate, living penalty, rewards, the activation functions and the network architecture. And the testing for both is done for various time intervals with its related accuracy value.

The combination of the environments or the learning techniques can be made possible which is the future scope of this paper.

2. SUPERVISED LEARNING MODEL

Supervised Learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. It infers a function from labeled training data consisting of a set of training examples.

In this case the images recorded from the car when ridden is the input data and the steering angle is the target label. And this supervised model in general is a technique that was implemented by the [1]NVIDIA's End-to-End Deep Learning for Self-Driving Cars. We have used the concept and modelled a system with different activation functions and different convolutional layers.

2.1 Implementation

The implementation is done in three main stages/phases:

a. Data Generation Phase

b. Training Phase

c. Testing Phase

2.1.1 Data Generation Phase

For the data generation part, where the data collected here is the images from the camera(s), which is nothing but the road that the car sees to its front and various physics variables like its speed, throttle, brake and steering. In order to get the data and for the environment we have used Udacity's open source version of the car simulator which comes with two pre-loaded environments and an efficient way to train and test the model.

In order to generate the data we select the training mode of the open source simulator which brings up a car with three camera's mounted on it, towards the left, right and center of the car. So we start moving the car with the keyboard inputs or with a joystick for smother transitions, we can now record this movement of the car which is controlled by us (a human driver) and this action will be mimicked by our model in the form of mapping images to actions.

This data can then be saved into a comma separated value file, which has the images from the camera's pointing to left, right and center of the car with the image's absolute path, the speed, throttle, brake and the steering values

2.1.2 Training Phase

The training process is inspired by the [1] NVIDIA's End-to-End Deep Learning for Self-Driving Cars, they have made use of an actual car fitted with three cameras on its left, right and center and had a human driver drive the car for 72hrs, recording the images from the cameras along with the car's speed, throttle, brake and the steering angle values.

We have mimicked the above process using the Udacity's open source version of the car simulator, by making a user (human driver) drive the car in the simulator using a keyboard for various laps and different intervals of time. And have recorded it with respect to the number of laps and various time intervals which is shown in the Table 1.

Table -1: Accuracy with respect to the no. of laps.

Number of Laps	Time taken to train the model	Accuracy of the model
1	15mins	20%
3	50mins	45%
5	3hrs	70%
10	8hrs	85%

In the model, we have used Keras for our deep learning framework with a sequential model. We start with doing image normalization and adding a 10 layer convolutional network layer with 5 fully connected layers to flatten the data. We have tried the model using various activation functions like the sigmoid activation function, rectified linear activation function and the exponential linear unit activation function, and found exponential linear unit the best activation function for this scenario as it takes care of the vanishing gradient problem.

Table -2: Accuracy with respect to the activation functions

Activation Function	Accuracy of the model
Sigmoid	70-85%
Rectified Linear Unit	80-90%
Exponential Linear Unit	85-95%

Now since the setup is ready, we try to find the data loss or the error that is by finding the difference between the predicted steering command and the actual steering command from the training data. Once we determine the mean of sum of squared error, that is then back propagated to the convolutional neural network which in turn produces a network computed steering value. And for optimization we have used Adam optimizer that does our Gradient decent function.

2.1.3 Testing Phase

A server-client structure is used for the testing, where the server is the open source simulator and the client, our python program.

Various dependencies and libraries like socketio which helps us work in terms of piping commands in real time using event handlers, Pillow for image manipulation and flask which is a web framework.

For testing, we start our simulator and then pass the model that was trained in the training phase to our testing script and simultaneously activate the autonomous mode in the simulator to which the program connects and outputs the required steering angle, speed, throttle and brake values to the autonomous car.

Here the prediction of the steering angle with respect to the image from the cameras is sent to the server and Thus giving us a self-driving car in a simulated environment.

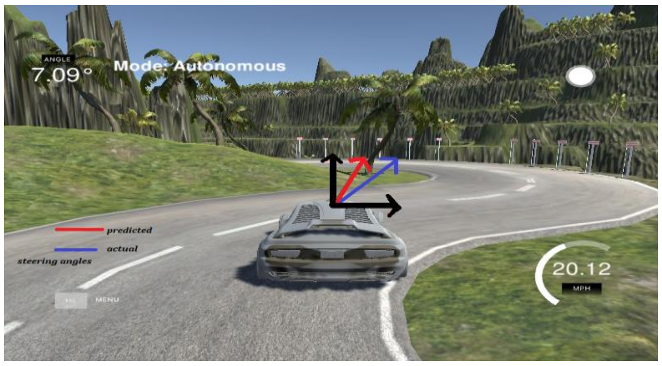

Fig -1: Testing the model

3. REINFORCEMENT LEARNING MODEL

Reinforcement learning is a type of Machine Learning algorithms which allows software agents and machines to automatically determine the ideal behavior within a specific context, to maximize its performance.

Reinforcement algorithms are not given explicit goals; instead, they are forced to learn these optimal goals by trial and error.

Here we are making use of the sole concepts of reinforcement learning like the Bellman Equation, Markov Decision Process, and Q Learning which is the base for Deep Q Learning Networks.

3.1 Concepts

3.1.1 Bellman Equation

Say we have a state s, an action taken by the agent as a, the reward R which the agent gets once it gets into a particular state and 'Υ (gamma) the discount.

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

The bellman equation assigns a value to a state by getting the maximum of all the actions possible with respect to the sum of the rewards that the agent gets in the current state for a certain action and the value of the following state or the state that the agent would end up in from the current state.

3.1.2 Markov Decision Process

The Markov Decision Process takes care of the non-deterministic search problem. That is the agent is bound the take any action in order to find its optimal policy or path and the actions it might take may not be in a deterministic way, thus we have a probability term that is multiplied to the value of the following state.

$$V(s) = \max_a\left(R(s,a) + \gamma \sum_{s'} P(s,a,s')V(s')\right)$$

3.1.3 Living Penalty

Generally an agent gets a reward say in our case a positive reward when it reaches its destinations and a negative reward when it goes off road, but a living penalty is a reward that is given to an agent throughout in its exploration phase so as to generally find an optimal path. The policy or the mapping changes with respect to the living penalty value that is assigned to the agent in its environment.

3.1.4 Q Learning

Q-learning is a simple way for agents to learn how to act optimally in controlled Markovian domains. It amounts to an incremental method for dynamic programming which imposes limited computational demands. It works by successively improving its evaluations of the quality of particular actions at particular states.

The value of the states are replaced by the Q functions or rather the Quality factor of the action in the respective states.

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

3.2 Design and Environment

We have made use of Kivy which is an open source Python library for development of our self-driving car interface or the environment and for the mapping purposes.

In the environment we have a car that goes from one destination to another, where in this case the destinations are predefined. As it moves it is encounters to many barriers and the car overcomes or surpasses these barriers and continue its route using the idea of Deep Q learning or in general Reinforcement learning technique(s).

The user is allowed to place or insert the barriers in the course of the car and the car surpasses these barriers in real time with minimal fault and training. The learning, with respect to this scenario is very fast as compared to the previously mentioned supervised learning model.

The possible actions that the car can take here is move left, right or straight. And the angle of the rotation to its left or right is specified as 10. The car is equipped with three sensors which are to its left, right and center and these sensors detect the barriers and avoids them by taking the opposite direction of the sensor that detects the barriers.

We also configure the rewards that the car gets when it hits the barrier as -1 which is the worst reward, the living penalty reward as -0.4, if it reaches its optimal path that is a path better than the previous one in terms of the distance taken it gets a reward of 0.2, a reward of -1 if it hits the environment window or the limit and a reward of 1 if it reaches its destination.

3.3 Implementation

In the reinforcement learning there is no need of explicit data generation because the data is obtained as the agent is exploring, and the training and testing happens simultaneously.

We are extensively using PyTorch deep learning framework for creating the neural network architecture. The neural network architecture is comprising of 2 fully connected layers in a linear manner with 30 nodes in the hidden layer, the first fully connected layer is with input and the hidden layer and the next is between the hidden layer and the number of actions the car can take.

Next we start building the feed forward network, for this we apply an activation function to our first fully connected layer and generate the Q values by passing the output of the activation function to the second fully connected layer. And return the Q values. This architecture can be improved by tweaking the number of nodes in the hidden layers and using different activation functions.

Then we continue to implement an experience reply for the agent as the Markov Decision process or the Q learning algorithm works with respect to analyzing a series of events and the consecutive states, that is the current state and the following state. The correlation among these states are more and the network will not be learning very well. So in order to avoid this we use experience reply which considers say about 100-1000 states and keeps them in the memory as the series of events so that an efficient decision can be taken by the agent and these bulk states are treated in various batches.

Finally we build our Deep Q learning model by passing various hyper parameters like the discount factor (Gamma), the model itself from the neural network architecture, the reply memory and the optimizer to do stochastic gradient decent where in this case an Adam Optimizer is used with the learning rate of 0.005 which gave the best result. We also determine the right action for the car to take that is whether to go left, right or straight using a softmax layer. Softmax gives a large probability to highest Q value. Softmax also takes care of the exploration of the environment by the agent really well. The learning is done in batches with respect to the reply memory technique. While learning we consider the maximum of the Q values of the next state as well as the reward obtained in the current state. We then calculate the loss using the smooth l1 loss (Huber loss - since it improves the Q learning) and pass the loss error to the optimizer to do the stochastic gradient decent and update the weights using the back propagation. Eventually an update is done with respect to the action and the state when the agent gets into a new state and a connection is made between the brain (the DQN logic) and the map function which is defined earlier in the section 3.2 Design and Environment.

3.4 Training and Testing

The training and testing can be visualized once we run the environment and the car, since the Deep Q Network (DNQ) logic is linked to the mapping part we need not explicitly pass any arguments to link them at the time of execution.

The car starts exploring the environment for some time and once it reaches its destination it gets a positive reward and it looks for its next destination checkpoint and continues to get positive reward from reaching checkpoints and finds an optimal path to do the same. But when we start placing barriers in the environment and when the car encounters these barriers it receives a negative reward, so in its next encounter it evades the barrier and becomes much efficient to its previous transition.

The best part about this system is that the barriers or the obstacles can be randomly anytime and anywhere on the environment and the car or the agent avoids or evades these barriers in real time.

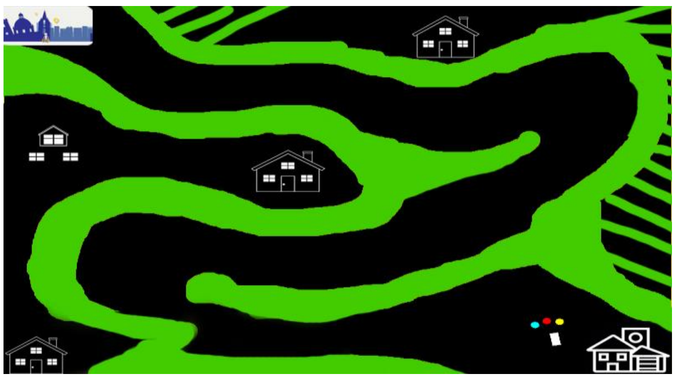

Fig -2: Bird's view of the environment

Thus, the agent updates the rewards and the scores continuously and takes the next actions with respect to these updated rewards, scores and the Q values.

This gives rise to a self-driving car using the reinforcement learning technique.

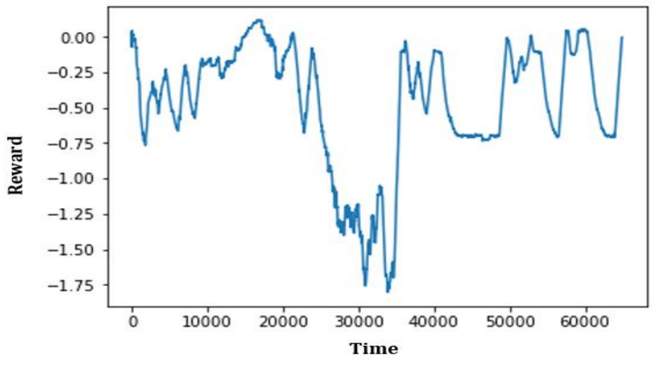

Chart -1: Behavior with respect to the time and the reward

4. SUPERVISED V/S REINFORCEMENT MODEL

In the supervised learning model the mapping between the images to the car's attributes like speed, steering angle, throttle and brake is done, so whenever the car/ model gets a new image it will act accordingly whereas in reinforcement learning model the car explores the environment at first for some time as a part of training and with respect to the rewards it gets, it moves in the environment taking various decisions like going left, right or straight.

Supervised learning model needs a lot of data and predefined training for a long time only then it can be implement accurately whereas reinforcement learning adapts quickly with less training and data.

Reinforcement learning model in this scenario can easily evade or surpass any obstacle thrown to it by the user anytime, anywhere whereas the supervised learning model fails to do so.

Supervised learning model can easily facilitate a large number of features and a high accuracy can be obtained but we will have to use a powerful enough computer that run on multiple graphics processing unit(s).

5. CONCLUSIONS

In this paper we have given an overview of the two different models that is the supervised learning model and the reinforcement learning model with respect to a self-driving car. And how each model behaves when the hyper parameters are tweaked, variations in the activation functions is made and how long the model is trained.

The model can be furthered improvised by changing the environments and the architecture, and the implementation in general can be improved by combining the learning models and further tweaking the hyper parameters or adding more layers to the architecture.

ACKNOWLEDGEMENT

We thank our family and friends for their priceless support and encouragement.

REFERENCES

[1] NVIDIA's End-to-End Deep Learning for Self-Driving Cars.

[2] The Theory of Dynamic Programming By Richard Bellman (1954)

[3] Markov Decision Processes: Concepts and Algorithms By Martijn van Otterlo (2009)

[4] Learning to Predict by the Methods of Temporal Differences BY Richard Sutton (1988)

[5] Reinforcement Learning I: Introduction By Richard Sutton.

[6] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. R. Howard, W. Hubbard and L. D. Jackel. Backprop-agation applied to handwritten zip code recognition.

BIOGRAPHIES

Athul Dev
Machine Learning and Data Science enthusiast.
***Experiences**:* Software Developer Intern at Power Grid Corporation of India, Machine Learning Intern at Latent Talent Technologies and Curl Analytics, Data Analyst Intern at CHR Solutions. Interested in Big Data Analytics, AI, ML and web development.

Roshni K S
IoT, embedded systems, microcontrollers, matlab and computer programming enthusiast.
***Experiences**:* Software Developer Intern at Power Grid Corporation of India, Power System Analyst Intern at KPTCL.