Exploring the Potential of Q-Learning Offers a Promising Pathway towards Achieving Artificially Intelligent Driving Capabilities

Chinmaya Nayak
Department of Research and Development,
Centre for Development of Telematics,
New Delhi-110030, India
chinmaya@cdot.in

***Abstract*—This research focuses on Artificially Intelligent driving techniques that are being used to train several machine-learning models to achieve complete human-like driving skills. Artificially Intelligent driving consists of training a machine to drive on a provided path to any vehicle (a car in this case) while simultaneously following all the traffic routes, providing passenger comfort and vehicle and passenger safety. In this research, since most of the available artificially intelligent driving models are set to work upon a predetermined path provided by the user and can only follow that path, I intend to further this model by providing a completely random path to the model and then evaluate its efficiency, the resources it requires to complete its whole path to training the model so that it can adapt to the randomly provided path with much faster speed and more accuracy as compared to traditional Artificially Intelligent vehicle driving models.**

***Index Terms*—Machine Learning, Reinforcement Learning, Q-learning, and Deep Learning**

I. Introduction

This dissertation is completely based on the mechanics of Reinforcement learning (RL). This is a branch of artificial intelligence that focuses on training agents to make decisions and operate in a way that will maximize the benefits that will accrue over time. It is based on the concept of learning through trial and error, where agents interact with the environment, receive feedback in the form of rewards, and learn to improve their decision-making strategies over time. RL algorithms employ exploration and exploitation techniques to find optimal policies that lead to the highest possible rewards. RL has diverse applications and has been successful in training agents to play games, control autonomous systems, and solve complex decision-making problems.[1], [2], [3], [4]

A. Machine Learning

One of the thesub-fieldss of artificial intelligence called machine learning involves teaching machines to recognize patterns in data and then use those patterns to forecast or make decisions. Machine learning is frequently used in real-world applications, including image identification, natural language processing, and predictive analytics. There exist various types of machine learning, but we will focus on only 3 see (Fig 1.1.

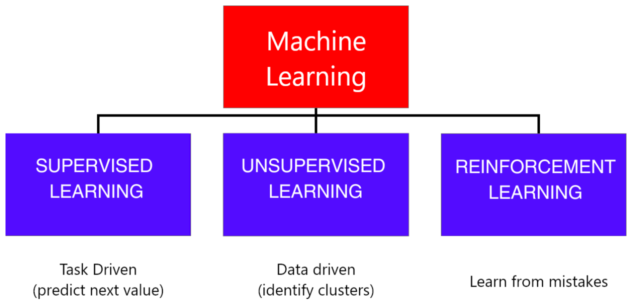


Fig. 1. Types of Machine Learning

- The most popular kind of machine learning, Supervised learning, involves developing an algorithm using a labeled data set. The intention is to train the algorithm to spot patterns in the data so that it can make precise predictions about future instances of unlabeled data. Examples of supervised learning include predicting the price of a house based on its features or classifying images of animals based on their characteristics.
- Unsupervised learning, on the other hand, is used when there is no labeled data available to the system. The goal is to determine or identify patterns or clusters in the data without any prior knowledge of what those patterns might represent. This type of learning is often used in exploratory analysis or anomaly detection.
- Reinforcement learning involves educating a computer algorithm to make choices based on data from its surroundings. The algorithm learns to take actions that maximize a reward signal while minimizing negative outcomes. This type of learning is often used in robotics, game playing, and autonomous vehicles..

In conclusion, machine learning is an effective tool for forecasting and making choices based on patterns discovered from data.

B. History of Reinforcement Learning

Since their creation in 2006, deep learning (DL) algorithms have been widely used by both researchers and businesses.

Since the spectacular victory over the ImageNet classification challenge in 2012, supervised deep learning has seen success after the triumph. Many academics are currently using this novel and potent family of algorithms to address a wide range of novel challenges, including how to effectively train intelligent behavior in reward-driven complex dynamic problems. Over here agent-environment interaction is defined through observation, Because learning environments have the Markov property, they can be thought of as Markov decision problems, which makes RL approaches possible Games had to be included in this group of environments. Inputs (the game world), actions (game controllers), and assessment criteria (game score) are often understood and simulated in a game-based environment. Classic RL algorithms from the 1990s could now tackle exponentially more complicated tasks, like games, over time, navigating across vast decision spaces, thanks to the development of DL and increased computer power. [2], [3], [5]

C. Introduction to Reinforcement Learning

A learning method similar supervised and unsupervised learning is reinforcement learning. Instead of learning from a labelled data set (or one that is not labelled), you learn from the errors that a reward system generates. We may say the agent gains knowledge through experience. To achieve an ideal state or goal, an agent (the car) see figure (Fig.2) interacts with an environment (the world). The agent acts in order to interact. .[6]

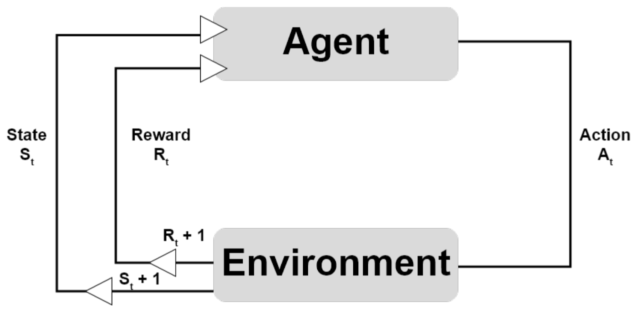


Fig. 2. Types of Machine Learning

- When the action is positive, the reward will be greater.
- When the action is negative like going on the sidewalk or hurting other cars then the reward will be lesser.

The goal of the agent is to choose the right action that gets the biggest reward.

- In this situation, the agent can predict future states and behaviors and decide which course of action to take right away to maximize potential rewards.
- Now we can calculate the total reward based on all rewards.
- When using reinforcement learning, you can create an environment and optimize the driving policy using a reinforcement learning algorithm.

D. Implementation of Reinforcement Learning

The implementation of Reinforcement Learning can be done in three ways: -

- Value-Based: The main goal of a value-based reinforcement learning strategy is to maximize a value function
- Policy-based: The objective of a policy-based RL approach is to create a policy where every action you do now will help you reap the greatest benefits in future. Two types of policy-based methods are:
- Deterministic: The policy results in the same action for any state.
- Stochastic: Every course of action has a chance of happening
- Model-Based: In this reinforcement learning method, a virtual model must be created for each environment. The agent acquires the abilities required to function in that setting.

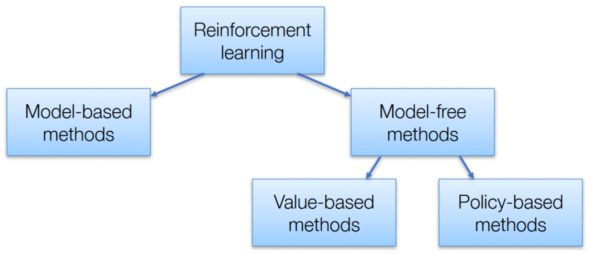


Fig. 3. Approaches to Reinforcement Learning

E. Q – Learning

Q-learning is a values-based learning algorithm in reinforcement learning Q-Values or Action-Values: Q-values are defined for states and actions. Q (S, A) is an estimation of the likelihood that taking action at state S would be good. As we will see in the following sections, the TD- Update rule will be used to iteratively compute this estimation of Q (S, A). Depending on the actions it does and the environment it interacts with, an agent transitions from one state to another multiple times during the duration of its existence, starting from a start state. Every time a transition occurs, the agent from the initial state acts, detects a reward from the environment, and then transitions to the following state. If the agent ever enters one of the ending states at any moment, there can be no more transitions see Fig (1.6). [7], [8], [9]

As seen below, the Temporal Difference or TD-Update rule: - This update rule to gauge the Q value is applied at each time step of the agent's interaction with the environment. The following defines the terminology used: S: Current state of the agent. A: The current action was chosen by a policy. S': The agent's final destination state. A: Choose the action with the highest Q-value in the following state using the most recent Q-value estimation as the next best action. R: The reward that is now being given as a result of the surroundings. The factor for Discounting Future Rewards. Future incentives must be

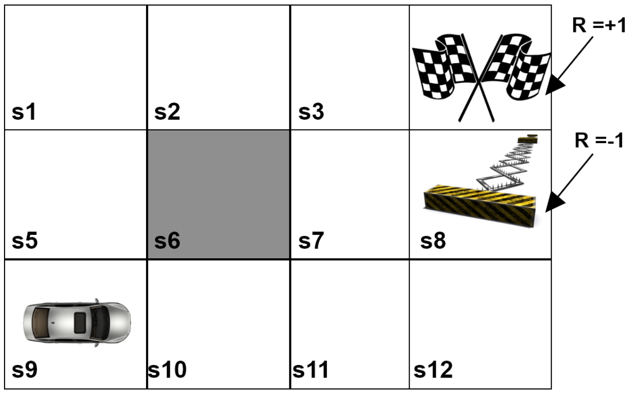


Fig. 4. Q – Learning

discounted since they are less valuable than current rewards. The discounting rule also applies in this case since the Q-value is an estimation of expected rewards from a state. Length of each update to the Q estimation (S, A).

F. Deep Q – Learning with Neural Networks

The Q-learning algorithm and deep neural networks are combined in Deep Q-Learning (DQL), a potent reinforcement learning method. In complicated situations with high-dimensional state spaces, it enables agents to learn the best course of action.

As it interacts with its surroundings during training, the agent gathers experiences in the form of state-action-reward-next state tuples and updates the Q-network with them. By reducing the difference between the target and predicted Q-values, which are calculated using a Bellman equation that takes into account the current reward and the maximum Q-value of the following state, the update is carried out. Exploration and exploitation should be balanced, DQL typically employs an exploration strategy, such as epsilon-greedy, which chooses random actions with a certain probability to encourage exploration of the environment. [10], [11], [12], [13], [14]

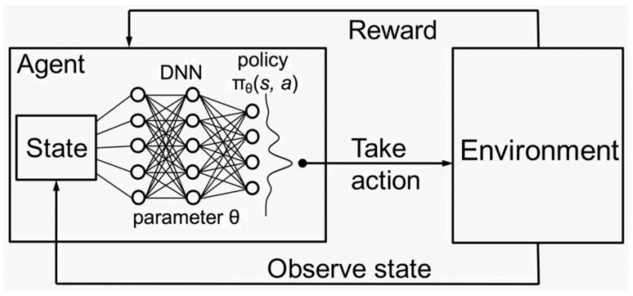


Fig. 5. Deep Q – Learning with Neural Networks

II. LITERATURE REVIEW

A branch of machine learning called reinforcement learning (RL) focuses on making decisions in challenging situations. A car with artificial intelligence can sense its surroundings and function on its own. There is never a need for a human driver or passenger, or even for someone to be inside the vehicle. Artificially intelligent vehicles are capable of doing everything a skilled human driver can accomplish while operating any place a conventional vehicle can go. Many applications of RL have emerged in recent years, including robotics, game playing, finance, and healthcare. A car with artificial intelligence can sense its surroundings and function on its own. There is never a need for a human driver or passenger, or even for someone to be inside the vehicle. The artificially intelligent vehicles are capable of doing everything a skilled human driver can accomplish while operating any place a conventional vehicle can go. Driverless cars can recognize objects, analyze circumstances, and make decisions based on object detection and object classification algorithms. Deep learning is a component of machine learning or the advancement of machine learning. Deep learning draws its inspiration from how the human brain processes information. To extract more exact information, it makes use of complex neural networks that continuously evaluate and learn from the incoming data. While supervised learning employs tagged training data, unsupervised learning uses less structured training sources. Either of one, supervised or unsupervised deep learning is possible. In contrast to machine learning, deep learning requires a large amount of computing power and training data to create more accurate results. These algorithms are inspired by the human brain, indicating that they acquire knowledge through their experiences. NVIDIA, a deep learning expert, claims that if a DNN is exposed to photos of a stop sign in various settings, it can learn to recognize stop signs on its own.[15], [16], [17], [18] Deep reinforcement learning is a new set of the latest and advanced algorithms that make use of batch calculations on graphics processing units (GPUs), reward/punishment costs, and the enormous computational power of modern machines (DRL). It was shown that neuroevolutionary methods might be used to analyze pixel data directly. The most notable advance in the age of DRL a year later was Google's creation of the Deep-Q-Network (DQN). This cutting-edge system might identify patterns in pixels in an unknowable environment and learn actions from them. However, early on there were several problems with neural networks serving as approximate methods (correlated inputs, fluctuating policies, huge gradients, etc.) that were eventually resolved by the organic growth of the larger DL field. For instance, a Machine Learning (ML) model's training process may be impacted by the correlation between its inputs, which frequently results in underfitting or overfitting. The use of numerous value functions or huge gradients by deducting a previously learned baseline has been used to solve other difficulties, such as policy degradation, which can result from value function overestimation. Trust region algorithms, such as relative policy optimization (PPO) or trust region policy optimization (TRPO), where the policy is changed, are other methods of addressing these instabilities. After testing with ATARI 26oo games at first, other trials with more difficult games were also conducted (DOTA2, Starcraft, Chess, Go, etc.). Last but not least, they showed that DQN-type algorithms could beat any conventional RL algorithm, defeating the paid professional human players who participated in the Atari game titles. DQL has demonstrated significant success in various domains, including playing Atari games and achieving superhuman performance. By employing deep neural networks, DQL can handle complex environments with high-dimensional input spaces and learn effective policies by approximating the Q-values.

III. IMPLEMENTAION OF DEEP – Q LEARNING

A. Working principle behind Deep Q – Learning

The working principle behind Deep Q-Learning (DQN) is to combine the Q-learning algorithm with deep neural networks to handle high-dimensional state spaces. The Q-values for each state-action pair are estimated by DQN using a deep neural network as a function approximator. The following explains the working principle of DQN:

- State and Action Representation: DQN represents states and actions using high-dimensional vectors or images. These representations capture the raw sensory inputs of the environment.
- DQN utilizes a deep neural network, often a convolutional neural network (CNN), as a function approximator to estimate the Q-values. The state representation is sent into the neural network; it then generates the Q-values for each possible action.
- Experience Replay: DQN incorporates experience replay, which entails putting the agent's experiences in a replay buffer (state, action, reward, and future state). A mini-batch of events is randomly selected from the replay buffer during learning. to decorrelate the training data and break the temporal dependencies between consecutive experiences.
- Q-Value Update: The DQN algorithm follows the Q-learning update rule to update the Q-values based on the observed reward and next state. However, instead of directly altering the Q-values after each step, DQN uses the neural network to Q-values for the current condition should be predicted. Selecting the action with the highest Q-value for action selection, following an exploration-exploitation strategy such as epsilon-greedy.
- Loss Function and Training: To determine the difference between the desired and desired Q-values, DQN applies a loss function. The goal Q-values are calculated by factoring in both the maximum predicted Q-value for the subsequent state and the immediate benefit. Gradient descent is used to train the neural network to reduce loss and increase the precision of the Q-value predictions.
- Exploration vs. Exploitation: DQN balances exploration and exploitation by using an epsilon-greedy policy. Initially, the agent explores the environment by taking random actions with a high exploration rate (epsilon). Over time, the exploration rate is gradually reduced to favor the exploitation of the learned Q-values.
- Iterative Learning: DQN iteratively repeats the process of taking actions, observing rewards and next states, updating the Q-values, and modifying the weights of the neural network. The neural network learns to estimate the ideal Q-values with different state-action pairs through this repeated process. DQN provides effective learning in high-dimensional state spaces by fusing using deep neural networks and the Q-learning algorithm. DQN has proved successful in resolving complicated reinforcement learning challenges, including playing Atari games, robotic control tasks, and more.

1) Algorithm of Deep Q – Learning: Algorithm of Deep Q – Q-Learning The algorithm behind Deep Q-learning (DQL) combines the Q-learning algorithm with Deep Neural Networks (DNN) to manage state spaces with huge dimensions. Here's a step-by-step explanation of the Deep Q-learning algorithm:

- Initialize replay memory: Create a replay memory buffer to store the agent's experiences. Experiences consist of tuples (state, action, reward, next state) and are collected during the agent's interaction with the environment.
- Initialize Q-network: Set up a deep neural network (often a convolutional neural network) as the Q-network. This network takes the state representation as input and outputs the Q-values for all possible actions.
- Initialize target network: Create a separate target network that has the same architecture as the Q-network but with frozen parameters. This network is periodically updated with the weights from the Q-network to provide stable target Q-values during training.
- Exploration vs. exploitation: Determine the action selection strategy. Initially, the agent explores the environment by taking random actions or using an exploration strategy like epsilon-greedy. As training progresses, the agent gradually shifts towards exploiting the learned Q-values more often.
- Repeat steps 6-10 for each episode or time step:
- Observe current state: Receive the current state from the environment.
- Action selection: Select a move made with the exploration-exploitation of a plan based on the current situation and the Q-network's predictions.
- Execute action and observe reward, next state: Apply the selected action to the environment, receive the reward, and observe the next state.
- Store experience: In the replay memory, store the experience tuple (state, action, reward, future state).
- Sample mini-batch and update Q-network: Pick a few memories from the replay memory to sample. Utilize the target network and the Q-learning revised rule to calculate the target Q-values.

Update the weights of the Q-network by minimizing the loss between the predicted Q-values and the target Q-values.

- Periodically update the target network: After a fixed number of steps or episodes, update the weights of the target network by copying the parameters from the Q-network.

- Repeat steps 5-11 until convergence or a predefined number of iterations.

When estimating the Q-values, a deep neural network can be used as a function approximator and give the agent the ability to work with high-dimensional state spaces is the basic idea behind DQN. Through the use of experience replay and a different target network, DQN stabilizes the learning process and improves the efficiency of Q-value updates. Over time, the DQN algorithm discovers how to approximate the ideal Q-values for different state-action pairs and enables the agent to make better decisions in complex reinforcement learning problems.

2) : sectionFlowchart of Reinforcement Learning The following describes the complete process flow of the reinforcement learning techniques followed universally along with a diagrammatic illustration see Fig (3.1).

- Start: Begin the reinforcement learning process.
- Initialize: Set up the initial state of the agent and the environment.
- Select Action: Use the current state and a policy (e.g., epsilon-greedy) to choose an action for the agent to take.
- Execute Action: Apply the selected action to the environment.
- Observe Reward and Next State: Receive a reward signal from the environment based on the action taken and observe the next state.
- Update Q-values: Use the observed reward and next state to update the Q-values of the current state-action pair using the Q-learning or deep Q-learning algorithm.
- Check for Termination: Determine if the episode or task has terminated based on specific criteria (e.g., reaching a terminal state, or exceeding a time limit).
- If Termination Condition Met: Proceed to step 11 (End). Otherwise, continue to step 9.
- Update State: Set the current state to the observed next state.
- Repeat Steps 3-9: Continue the process by selecting a new action based on the updated state.
- End: Terminate the reinforcement learning process.

3) Dataflow diagram of Reinforcement Learning: This data flow diagram represents the flow of information and control between the different steps involved in the reinforcement learning process. The arrows indicate the direction of data flow, and the rectangles represent the steps or processes. The following are the features and methods that can be associated with each step:

- Initialize: This step involves setting up the initial state of the agent and the environment. Relevant features could include initializing the agent's state variables and the environment's initial conditions.
- Select Action: This step involves choosing an action for the agent to take based on the current state and a policy. Relevant features could include the state representation, the policy used (e.g., epsilon-greedy), and methods for action selection.

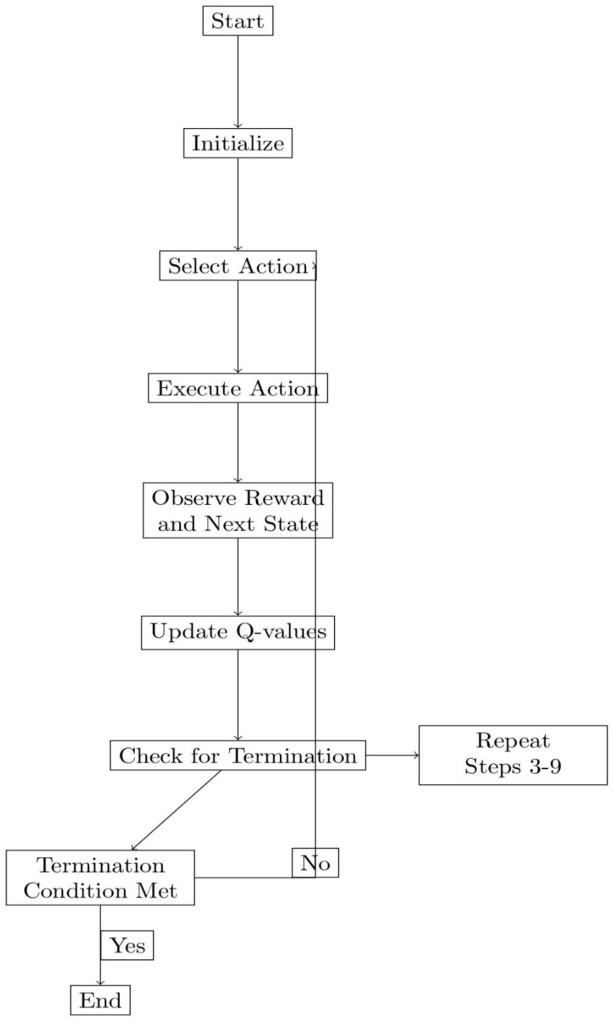


Fig. 6. Flowchart of Reinforcement Learning

- Execute Action: This step applies the selected action to the environment. Relevant features could include the action taken by the agent and the methods for applying the action to the environment.
- Observe Reward and Next State: This step involves receiving a reward signal from the environment and observing the resulting next state. Relevant features could include the reward value received and the observed next state.
- Update Q-values: This step updates the Q-values of the current state-action pair based on the observed reward and next state. Relevant features could include the Q-value table or function, the update algorithm (e.g., Q-learning, deep Q-learning), and the methods for updating the Q-values.
- Check for Termination: This step determines if the episode or task has terminated based on specific criteria. Relevant features could include the termination conditions to be checked and the methods for evaluating these

conditions.

- Update State: This step updates the current state to the observed next state. Relevant features could include the state variables and the methods for updating the state.

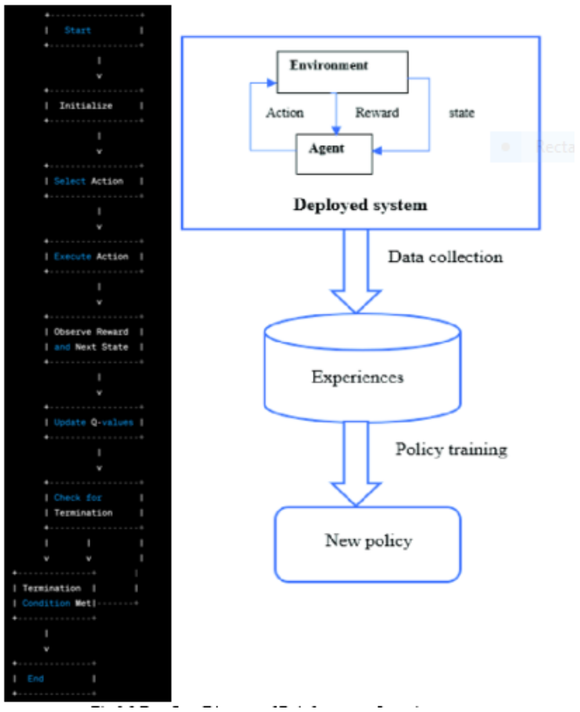


Fig. 7. Dataflow Diagram of Reinforcement Learning

IV. PROPOSED MODEL

A. *Working Principle of the proposed model*

In this work, we want to run our trained model on custom tracks and simultaneously achieve a higher accuracy in a short amount of iterations/times. Due to the higher complexity of neuro evolution techniques which are most preferred over other supervised learning techniques and also due to the difficult implementation of the above technique in Python code, we are going to provide a simpler and more precise solution through the concepts of Q- Q-learning and further contribute to the problem by working upon Deep Q-Learning techniques in Python. Also, we are going to run our models on multiple test cases to ensure higher fitness and accuracy.

The following are the objectives of this research work: -

- Implementation of deep q-learning technique in this research work.
- To achieve maximum efficiency, with a given throughput and least delay possible.
- To be able to complete a user-given track with no collision and provide better accuracy.

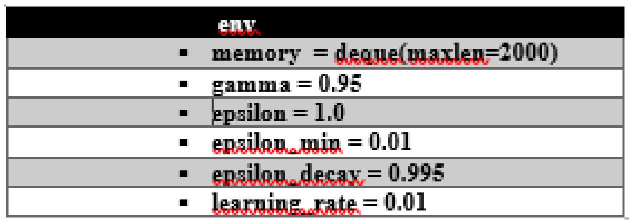
This research work focuses on an intelligent driving technique that is being used to train several machine-learning models to achieve complete human-like driving skills. Artificially Intelligent driving consists of training a machine to drive on a provided path to any vehicle (a car in this case) while simultaneously following all the traffic routes, providing passenger comfort and vehicle and passenger safety. In this research, since most of the available artificially intelligent driving models are set to work upon a predetermined path provided by the user and can only follow that path, I intend to further this model by providing a completely random path to the model and then evaluate its efficiency, the resources it requires it to complete its whole path to training the model so that it can adapt to the randomly provided path with much faster speed and more accuracy as compared to traditional Artificially Intelligent vehicle driving models.

- Train the pre-existing model to be able to drive on a random path as quickly as possible utilizing as little time and resources as possible.
- To provide better accuracy for the newly trained model using deep q-learning techniques and supervised reinforcement learning models.
- To provide a well-trained model that can greater throughput and better efficiency with a certain amount of accuracy.

This research work focuses on intelligent driving techniques that are being used to train several machine-learning models to achieve complete human-like driving skills. Artificially Intelligent driving consists of training a machine to drive on a provided path to any vehicle (a car in this case) while simultaneously following all the traffic routes, providing passenger comfort and vehicle and passenger safety. In this research, since most of the available artificially intelligent driving models are set to work upon a predetermined path provided by the user and can only follow that path, I intend to further this model by providing a completely random path to the model and then evaluate its efficiency, the resources it requires it to complete its whole path to training the model so that it can adapt to the randomly provided path with much faster speed and more accuracy as compared to traditional Artificially Intelligent vehicle driving models. The process involved in this proposed model and its implementation is divided into the following sub-sections discussed below: -

- Data Collection – This is the initial or the first step which requires the collection of data from multiple sources such as websites, articles, research papers, and journals. This data should be relevant to the solution that the aforementioned model is created for. It should be of a higher quality and properly formatted so that it can be processed easily by the enforced algorithm. In this project, I am going to use the following data for creating my model. An Environment – I am going to create the environment in the Pygame library used in Python programming language. The Agent – For applying reinforcement learning, I am going to use a small image of a car that is going to be my agent and an image of a track on which the model is going to be trained. The Model – The model that is going to be used will be created using the Keras library used in Python programming language.

- Data Preprocessing - This is the second step in the deep learning process. It involves cleaning the data and preparing it for use in the deep learning algorithms. This includes removing any irrelevant data, normalizing the data, and transforming the data into a format that can be used by the algorithms. Data preprocessing is an important step as it ensures that the data is in the correct format and contains no errors.
- Model selection is the third step in the deep learning process. It involves selecting the appropriate deep-learning model for the problem at hand. This includes selecting the type of model, such as a convolutional neural network or a recurrent neural network, as well as the parameters of the model. The model should be chosen based on the data and the problem that needs to be solved.
  For this research project, I am going to use an h5 model created in the Keras library that contains multiple learning biases, approximation functions, and optimizing functions as well.
- Model training is the fourth step in the deep learning process. It involves training the model on the data that has been collected and preprocessed. This includes optimizing the model parameters to ensure that the model can accurately predict the output given the input data. The model should be trained until it can accurately predict the output given the input data. This project requires comprehensive working principles of the Keras library which comprises the use of hyperparameters in Python Integrated Environment. The hyperparameters include: -



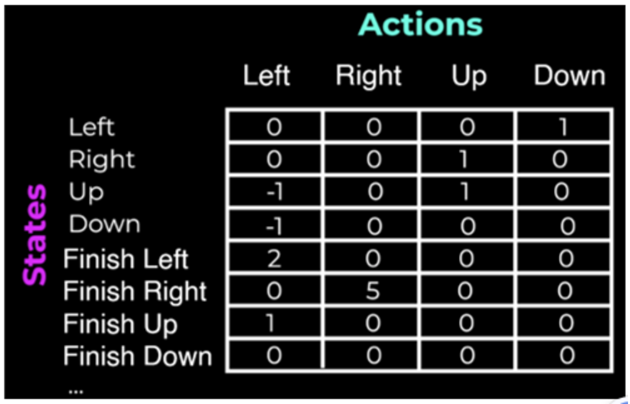
env
- memory = deque(maxlen=2000)
- gamma = 0.95
- epsilon = 1.0
- epsilon_min = 0.01
- epsilon_decay = 0.995
- learning_rate = 0.01

Fig. 8. Hyperparmeters used

- Model evaluation is the fifth step in the deep learning process. It involves evaluating the performance of the model on the data that has been collected and preprocessed. This includes measuring the accuracy of the model on the data and comparing it to other models. The model should be evaluated to ensure that it can accurately predict the output given the input data. The model evaluation is discussed in Chapter 5 which also contains the the measurement of accuracy of the proposed model.
- Model deployment is the sixth step in the deep learning process. It involves deploying the model in a production environment. This includes setting up the environment, such as the hardware and software, and deploying the model. The model should be deployed in a secure environment to ensure that it is not compromised. Once the model is deployed, it should be monitored to ensure that it is performing as expected. The model deployment is also discussed in Chapter 5 along with other observations and results.

In this work, we want to run our trained model on custom tracks and simultaneously achieve a higher accuracy in a short amount of iterations/times. Due to the higher complexity of neuro evolution techniques which are most preferred over other supervised learning techniques and also due to the difficult implementation of the above technique in Python code, we are going to provide a simpler and more precise solution through the concepts of Q- Q-learning and further contribute to the problem by working upon Deep Q-Learning techniques in Python. Also, we are going to run our models on multiple test cases to ensure higher fitness and accuracy.

B. Algorithm of the proposed model



States	Actions: Left	Right	Up	Down
Left	0	0	0	1
Right	0	0	1	0
Up	-1	0	1	0
Down	-1	0	0	0
Finish Left	2	0	0	0
Finish Right	0	5	0	0
Finish Up	1	0	0	0
Finish Down	0	0	0	0
...				

Fig. 9. The Q Table

1) The Q-Learning Algorithm: This is a value-based approach based on a Q-Table The Q-Table provides the largest expected future payoff for each activity at each state. We may then select the action with the highest reward using this Q-Table. see Fig (4.1). Here we want to teach an AI how to play race a car in a game environment. In this game, the car tries to reach and cross a circuit/track without hitting the wall or going in reverse. The activities and states can be listed in a Q-Table. The car's four possible maneuvers—turning left, right, up, and down will be represented by columns. The present direction, including left, right, up, and down, can also be the state. The rows are as shown. To further characterize the current condition, we can add further states. For instance, we may mention where the goal lines are and add the state's goal or the car's right, up, or down barriers. We could provide more information about the walls and their status by doing so, but we'll omit this for the sake of simplicity. We learn more about the environment as we provide more state information, but our system also becomes more complex. The values of the rows and columns, as well as every single cell, will represent the maximum predicted future reward for the specified state and activity. We refer to this as the Q-value.

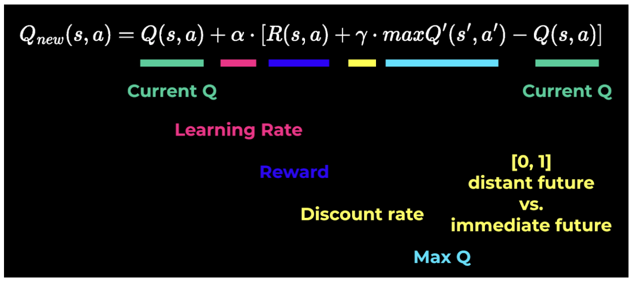
Q Learning Algorithm The calculation of the Q value is not done in a predetermined way. Instead, we approach the Q-improvement table iteratively. This process is known as training or learning. This is how the Q-Learning algorithm functions:

- 1. First, Initialize each Q-value, with 0.
- 2. Then, choose an action-a in the current state-s based on the current best Q-value.
- 3. After step 2, perform this action and observe the outcome (new states).
- 4. Then, calculate the reward R following this action.
- 5. Finally, update Q with an updated formula known as the Bellman Equation.

Repeat steps 2 through 5 until learning no longer advances, at which point we should have a useful Q-Table. The Q-Table can therefore be used as a reference sheet that always indicates the optimum course of action for a particular state.

Understanding Exploration vs. Exploitation trade-off To allow the agent to investigate the surroundings, we initially choose the action at random. To ensure that the agent uses the information it has, we reduce random exploration as we gain more training steps and increase exploitation. This happens when the Q values are all zeroes. A parameter commonly referred to as the epsilon () parameter regulates this trade-off in the calculations.

Reward Here, we're working to create a mechanism for the game's rewards. In the instance of this car game, we can award 1 point if the vehicle crosses a goal line, -1 point if it hits a wall or drives backward, and 0 points for all other permissible maneuvers. Now that we have all these components, we can use the Bellman equation: As seen in Fig (4.2) we can modify our Q value as follows: -



$$Q_{new}(s,a) = Q(s,a) + \alpha \cdot [R(s,a) + \gamma \cdot maxQ'(s',a') - Q(s,a)]$$

Fig. 10. Bellman equation

where, Current Q – represents the estimated value of taking a particular action in a given state, considering both the immediate reward and the expected future rewards. Learning Rate – determines the extent to which new information overrides the existing Q-value estimate during the learning process. Reward – quantifies the immediate desirability or value of being in a particular state and taking a specific action. Discount Rate – determines the weight given to future rewards, allowing for the consideration of long-term consequences in decision-making. Max Q – selects the maximum Q-value among all possible actions in a given state, representing the optimal expected value for making the best decision.

The rate of discount, which ranges from 0 to 1, expresses how much an agent is driven by rewards in the future as opposed to those that would be received right away. Finally, utilizing this iterative learning technique or model, we can produce an excellent Q-Table.

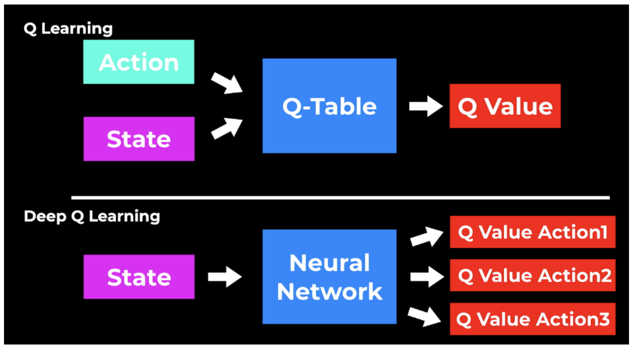
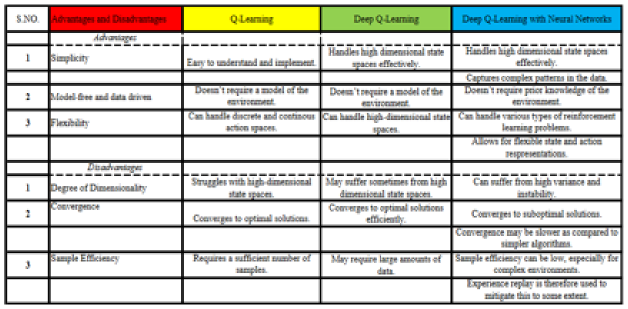


Fig. 11. Q-Learning vs Deep Q-Learning with Neural Network

Understanding the common differences between Q learning, deep Q learning, and deep Q network. Table (2.) There exists is a very slight distinction only. Q-learning is a reinforcement learning algorithm that helps to solve sequential tasks. It does not need to know how the world works (it's model-free) and it can learn from past experiences including from different strategies (so it is off-policy). It tries to predict a value that reflects a future expected reward which indicates how good any possible action is in a given state. After training, an agent can then just take the action that indicates the highest reward at every step to maximize its reward over time In the past, Q-learning was limited to very small state and action spaces because the computational requirements made more complex problems impractical to deal with. The Q-values were updated at every step in a Q-table with a row for every state and a column for every action.



S.NO.	Advantages and Disadvantages	Q-Learning	Deep Q-Learning	Deep Q-Learning with Neural Networks
	Advantages			
1	Simplicity	Easy to understand and implement.	Handles high dimensional state spaces effectively.	Handles high dimensional state spaces effectively.
				Captures complex patterns in the data.
2	Model-free and data driven	Doesn't require a model of the environment.	Doesn't require a model of the environment.	Doesn't require prior knowledge of the environment.
3	Flexibility	Can handle discrete and continous action spaces.	Can handle high-dimensional state spaces.	Can handle various types of reinforcement learning problems.
				Allows for flexible state and action respresentations.
	Disadvantages			
1	Degree of Dimensionality	Struggles with high-dimensional state spaces.	May suffer sometimes from high dimensional state spaces.	Can suffer from high variance and instability.
2	Convergence	Converges to optimal solutions.	Converges to optimal solutions efficiently.	Converges to suboptimal solutions.
				Convergence may be slower as compared to simpler algorithms.
3	Sample Efficiency	Requires a sufficient number of samples.	May require large amounts of data.	Sample efficiency can be low, especially for complex environments.
				Experience replay is therefore used to mitigate this to some extent.

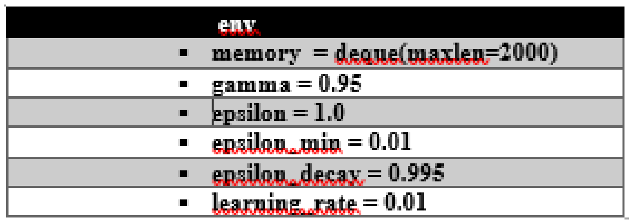
Fig. 12. Comparison Table

V. Result Analysis and Implementation

A. *Setting Up the Game Environment*

The following images depict the current progress of this research work also I have been able to map out a complete closed circuit/lap through basic means of design and model implementation. This project requires comprehensive working principles of the Keras library which comprises the use of hyperparameters in Python Integrated Environment. This project requires comprehensive working principles of the Keras library which comprises the use of hyperparameters in a Python Integrated Environment. The hyperparameters include: -

The following images depict the current progress of this research work also I have been able to map out a complete closed circuit/lap through basic means of design and model implementation. This project requires comprehensive working principles of the Keras library which comprises the use of hyperparameters in Python Integrated Environment. This project requires comprehensive working principles of the Keras library which comprises the use of hyperparameters in Python Integrated Environment. The hyperparameters include: -



env
▪ memory = deque(maxlen=2000)
▪ gamma = 0.95
▪ epsilon = 1.0
▪ epsilon_min = 0.01
▪ epsilon_decay = 0.995
▪ learning_rate = 0.01

Fig. 13. Hyper parameters used

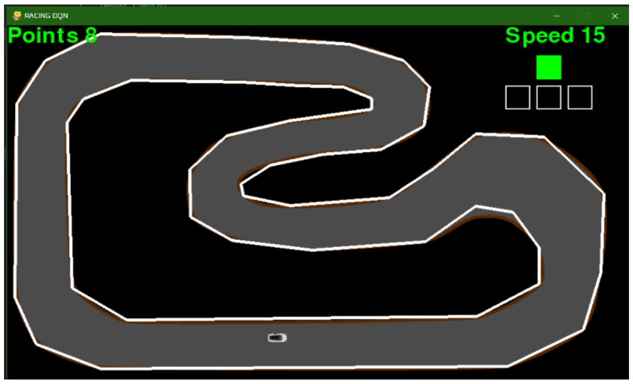


Fig. 14. The Game Environment

This image represents a basic layout of the track. The points represent several iterations or movements in the forward or backward direction of the car. The keys represent the direction of the car and track its actions.

1) Defining Actions and Goals: GOAL REWARD = 1 LIFE REWARD = 0 PENALTY = -1/* The above code represents what action will the car perform using DQN. This decision is purely based upon the provided epsilon parameter that is dependent upon a random action and some previous prediction based on the model's past training.

2) Movement and Distance Sensing: This image represents the sensing lines that the car has got through DQN to move to the furthest distance possible to be rewarded and calculate its actions accordingly.

the movement of the car throughout the closed circuit/track. The car here basically can travel in any of the desired directions as shown in Fig 6. Here the car takes one step and

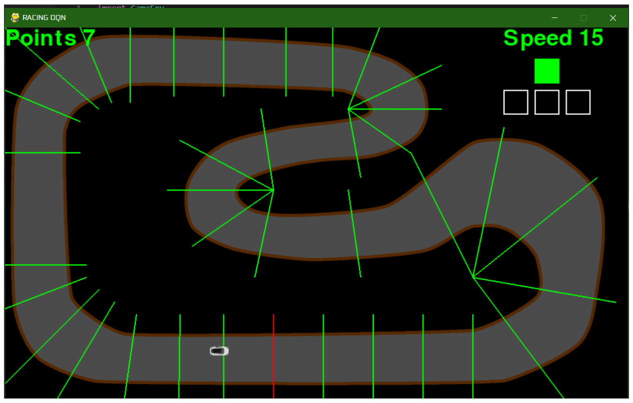


Fig. 15. Goals and Actions

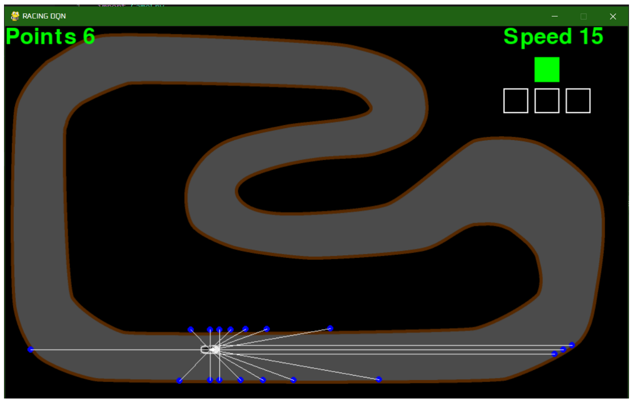


Fig. 16. Movement and Sensing

then calculates the distance between the next wall in its path. It then selects the fastest path it can travel in a single iteration and moves accordingly. As shown in Fig 6. the car can even move diagonally on its path but it senses the farthest point it can see to move that distance in a single step.

B. Results and Observations

Phases of training before

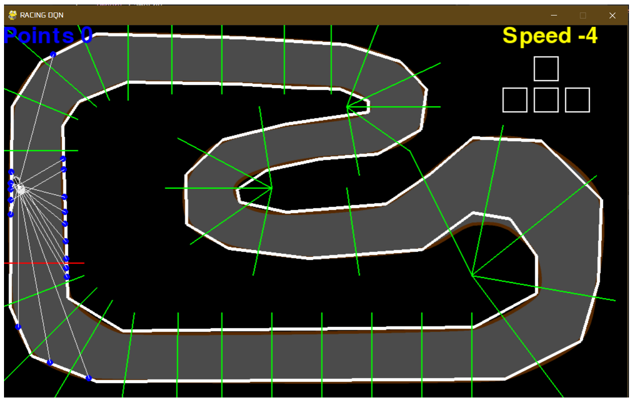


Fig. 17. Before Training

As seen in the above Fig7, the car is not able to traverse a single step at the beginning itself and hence has crashed into the wall. Also, we can see that the car has gone in reverse as indicated by the negative speed measure and hence has gained no point. After leaving the car and programming running for at least half an hour I was able to achieve some movement shown in the next figure. Here the car is simply learning its first steps and trying to learn not to make the same mistake again and again. After leaving the program running for some time

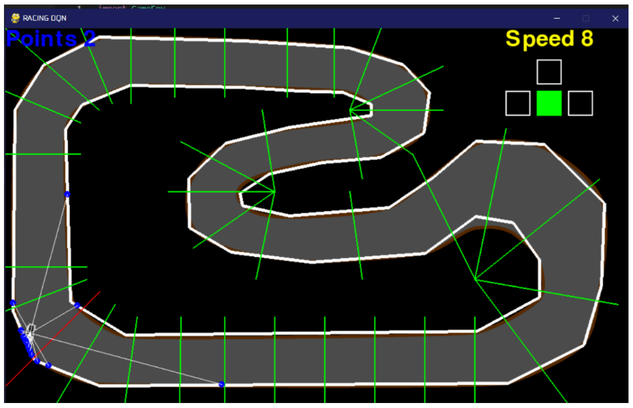


Fig. 18. Before Training

(approx. an hour), the car showed some movement and didn't crash at the same point, instead it, moved past the previous crashing point calculated the best distance in front of it, and moved accordingly. But here also when it came around the turn for the very first time it crashed. By repeating the same steps again and again, I am going to train this model so that it moves along the track and achieves a target goal without colliding with any of these walls.

After training phase After completing a certain amount of episodes the model was able to continue farther and farther each time going a bit longer than the previous episode. This process is shown in the observation images below. As discussed above by repeating the same steps again and again, after each episode the model got better and showed visible improvements across various parameters.

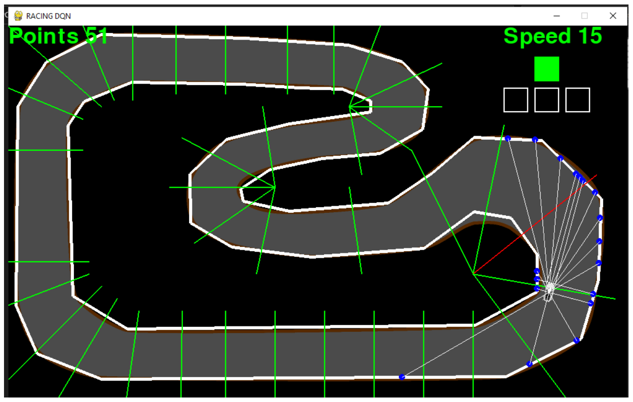


Fig. 19. Points Gained

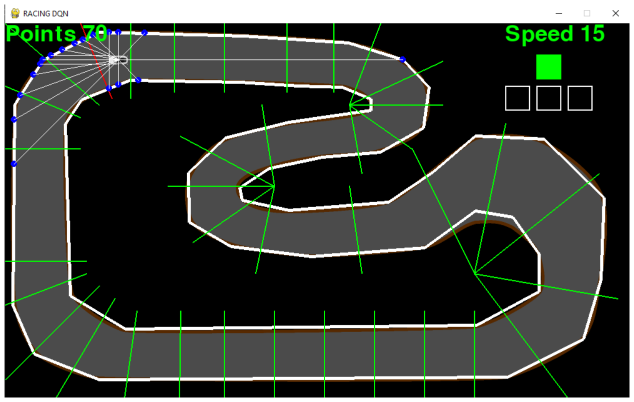


Fig. 20. Points Gained

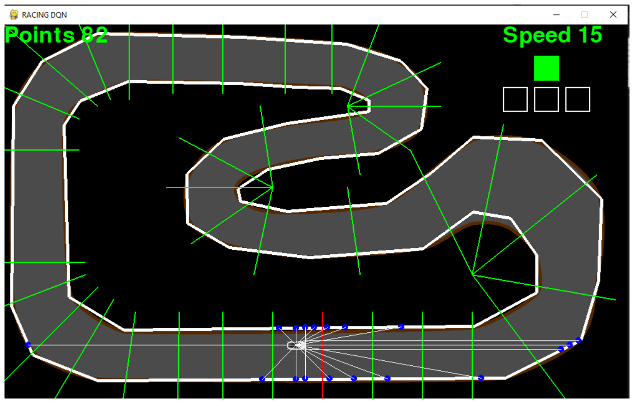


Fig. 21. Points Gained

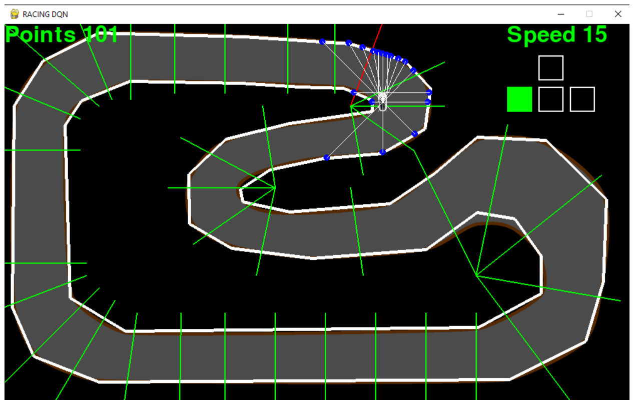
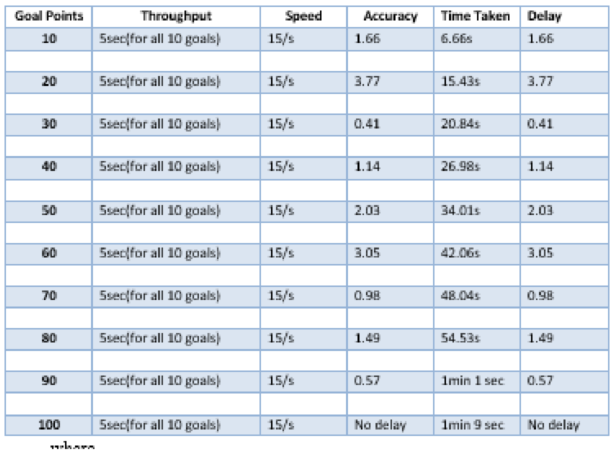


Fig. 22. Points Gained

As seen in the above images, we can see that the model has been trained with much better results as compared to before the training was complete. All the observations and results have been mapped on the next page in tabular data and also visualized as a graphical diagram for better understanding and representation. where, Goal Points – No. of goal line, the agent hast to cross as part of the reward for positive reinforcement. Throughput – Total time taken to complete a single lap or cover a complete track. Speed – Speed at which the agent



Goal Points	Throughput	Speed	Accuracy	Time Taken	Delay
10	5sec(for all 10 goals)	15/s	1.66	6.66s	1.66
20	5sec(for all 10 goals)	15/s	3.77	15.43s	3.77
30	5sec(for all 10 goals)	15/s	0.41	20.84s	0.41
40	5sec(for all 10 goals)	15/s	1.14	26.98s	1.14
50	5sec(for all 10 goals)	15/s	2.03	34.01s	2.03
60	5sec(for all 10 goals)	15/s	3.05	42.06s	3.05
70	5sec(for all 10 goals)	15/s	0.98	48.04s	0.98
80	5sec(for all 10 goals)	15/s	1.49	54.53s	1.49
90	5sec(for all 10 goals)	15/s	0.57	1min 1 sec	0.57
100	5sec(for all 10 goals)	15/s	No delay	1min 9 sec	No delay

Fig. 23. Observations and Results

will move across the environment. Accuracy – Time taken – Time taken to complete a specific no. of goals defined by the user. Delay – (Time taken for the current episode – Time taken for the previous episode - Throughput)

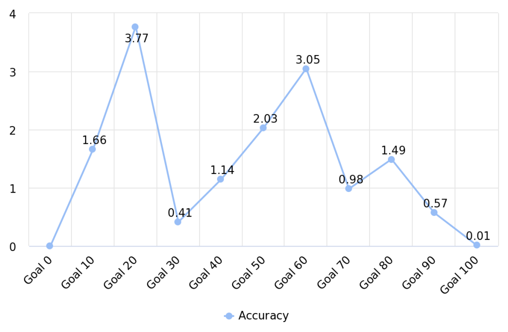


Fig. 24. Points Gained

As we can see from the above graph goal on (0,0) represents a 100 per accuracy, as we iterate with the value of 10 goals our accuracy graph shows some increases and decreases in the form of accuracy level but at the last of 100 goals it reached to the accuracy level of 99.8per. Hence, the model had been trained with a much better accuracy

VI. Conclusion

This research work concludes that this reinforcement learning technique trains a given system with a much better and an efficient model training when used with the proposed model. The objectives of this research were met at full capacity. The challenges faced during this research work were resolved to some extent. Finally, the goals and the results obtained from the performance analysis of the proposed model were achieved with better accuracy and efficiency.

The implementation of the proposed model has been done in a Python IDE for the time being. Seeing the endless possibilities, we can further emulate this model in a 3D environment having various constraints and test cases to increase the model's usability and real-world implementation for Intelligent Driving Systems. At the current stage, the model is only executable in a basic layout with the fundamental working principle of Deep Q-learning strategies. In further stages, the proposed model can be updated with the use of many other Python-based libraries to increase the currently trained model's accuracy and efficiency such as delay and throughput with the possible factor of elimination of collision with other objects more frequently.

References

[1] L. Garcia Cuenca, E. Sanz, J. Fernández, and N. Aliane, "Autonomous driving in roundabout maneuvers using reinforcement learning with q-learning," *Electronics*, vol. 8, p. 1536, 12 2019.

[2] H. Pradana, M.-S. Dao, and K. Zettsu, "Augmenting ego-vehicle for traffic near-miss and accident classification dataset using manipulating conditional style translation," in *2022 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, 2022, pp. 1–8.

[3] Z. Li, K. W. E. Cheng, and K. K. W. Chan, "An overview of factors influencing the mass adoption of self-driving vehicles," in *2022 IEEE 9th International Conference on Power Electronics Systems and Applications (PESA)*, 2022, pp. 1–5.

[4] C.-C. Chen, Y.-H. Guan, N. R. Novianda, C.-C. Teng, and M.-H. Yen, "Real-time traffic sign detection for self-driving and energy-saving driving based on yolov4 neural network," in *2022 IEEE/ACIS 23rd International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, 2022, pp. 208–211.

[5] M. A. Pandya, P. Siddalingaswamy, and S. Singh, "Explainability of image classifiers for targeted adversarial attack," in *2022 IEEE 19th India Council International Conference (INDICON)*, 2022, pp. 1–6.

[6] L. Jia, H. Zhong, X. Wang, L. Huang, and Z. Li, "How do injected bugs affect deep learning?" in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2022, pp. 793–804.

[7] A. Rangesh, B. Zhang, and M. M. Trivedi, "Gaze preserving cyclegans for eyeglass removal and persistent gaze estimation," *IEEE Transactions on Intelligent Vehicles*, vol. 7, no. 2, pp. 377–386, 2022.

[8] A. Rangesh, N. Deo, R. Greer, P. Gunaratne, and M. M. Trivedi, "Autonomous vehicles that alert humans to take-over controls: Modeling with real-world data," in *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, 2021, pp. 231–236.

[9] S. Q. Zulqarnain and S. Lee, "An efficient driver selection algorithm for controlling multiple vehicles in remote driving," in *2021 International Conference on Information Networking (ICOIN)*, 2021, pp. 20–23.

[10] S. Khanal, K. Thar, and E.-N. Huh, "Route-based proactive content caching using self-attention in hierarchical federated learning," *IEEE Access*, vol. 10, pp. 29 514–29 527, 2022.

[11] S. Khanal, K. Thar, M. D. Hossain, and E.-N. Huh, "Proactive content caching at self-driving car using federated learning with edge cloud," in *2021 Twelfth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2021, pp. 129–134.

[12] L. Kumar, D. Choudhury, A. R. Paduri, S. Kumar, D. Sahoo, J. Murthy, and N. Darapaneni, "Electric vehicle (ev) preventive diagnostic system: Solution for thermal management of battery packs using aiot," in *2023 IEEE 13th Annual Computing and Communication Workshop and Conference (CCWC)*, 2023, pp. 0041–0046.

[13] A. Ndikumana, N. H. Tran, D. H. Kim, K. T. Kim, and C. S. Hong, "Deep learning based caching for self-driving cars in multi-access edge computing," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 5, pp. 2862–2877, 2021.

[14] H. Abualsaud, S. Liu, D. B. Lu, K. Situ, A. Rangesh, and M. M. Trivedi, "Laneaf: Robust multi-lane detection with affinity fields," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 7477–7484, 2021.

[15] B. Jang, M. Kim, G. Harerimana, and J. Kim, "Q-learning algorithms: A comprehensive classification and applications," *IEEE Access*, vol. PP, pp. 1–1, 09 2019.

[16] T. Hammerbacher, M. Lange-Hegermann, and G. Platz, "Including sparse production knowledge into variational autoencoders to increase anomaly detection reliability," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, 2021, pp. 1262–1267.

[17] M. Al-Zeyadi, J. Andreu-Perez, H. Hagras, C. Royce, D. Smith, P. Rzonsowski, and A. Malik, "Deep learning towards intelligent vehicle fault diagnosis," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–7.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.