DS595/CS525-RL Project Report

Tactical Decision-Making for Autonomous Driving

<u>Group 5</u>
Varun Eranki
Akshay Sadanandan
Sayali Shelke
Manasi Shrotri

Table of Contents:

1. Introduction

Using deep neural networks, it is possible to find and localize certain objects (e.g. cars, pedestrians, cyclists, or traffic signs etc.) on the images, which is an important milestone in the development of self-driving vehicles. This project gathers a collection of environments for decision-making in Autonomous Driving. Our goal is to train agents that can drive through a ‘highway’ environment safely using different observation and action spaces on different algorithms and evaluate which is the best way to observe an environment for autonomous vehicles.

2. Literature Review

Highway Environment Model for Reinforcement Learning

Aim: This paper presents, highway simulation model. The environment is based on the Open AI Gym framework. The framework simulates highway environment with configurable traffic flow and ideal sensor models.

The environment presented in this paper has which includes three different models:
1. **Vehicle Model:** The main parameters of the model are: L-axle length and m weight. State parameters are x, y vehicle position, v longitudinal speed and raw angle. Control parameters are the acceleration demand, and the steering angle.

2. Environment sensing: This has common automotive sensors, such as camera and radar-based systems. This way the environment information that is available for the agent is its position and heading relative to the highway lane, and also the relative states of the vehicles in the ego and adjacent lanes.

3. **Traffic generation and behavior for environmental vehicles:** Vehicles are generated to reach the desired traffic density, though afterwards the simulation keeps on going to produce a random situation.

Reinforcement learning method:
Vanilla policy gradient method was used to train the model. The aim of training was to minimize loss function. In this model, policy function was multilayered neural network
Training:
For training the model. Different combinations of steering and acceleration values were used. For the reward multiple rewards were considered. 17 total steps were considered with 25 number of actions (combination of steering and acceleration)
Results:
Avg reward of the last 100 episodes converged to 480 in 25k episodes.

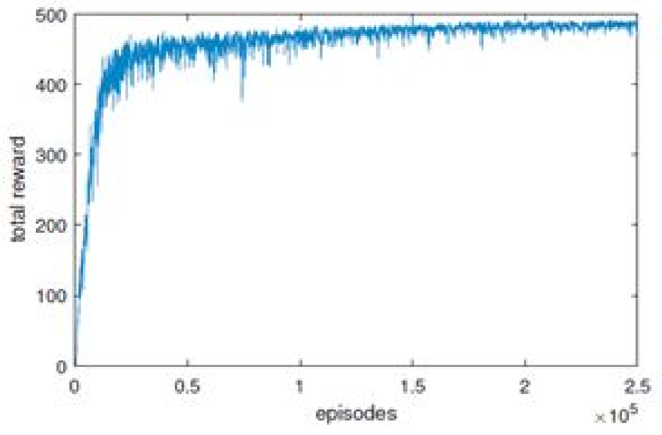


Figure 1: Plot of reward vs number of episodes

While evaluating this model, the average reward value achieved was 478.9, and there were 9 front and 13 rear collisions. The agent also kept the desired speed of 130 km/h and changed lanes.

3. Environment Description

The observations on the environment can be set on numerous parameters such as duration of every episode, number of lanes on the highway, speed range of all the vehicles, number of vehicles in an episode etc.

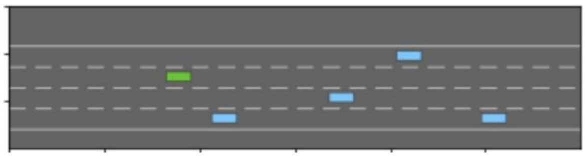
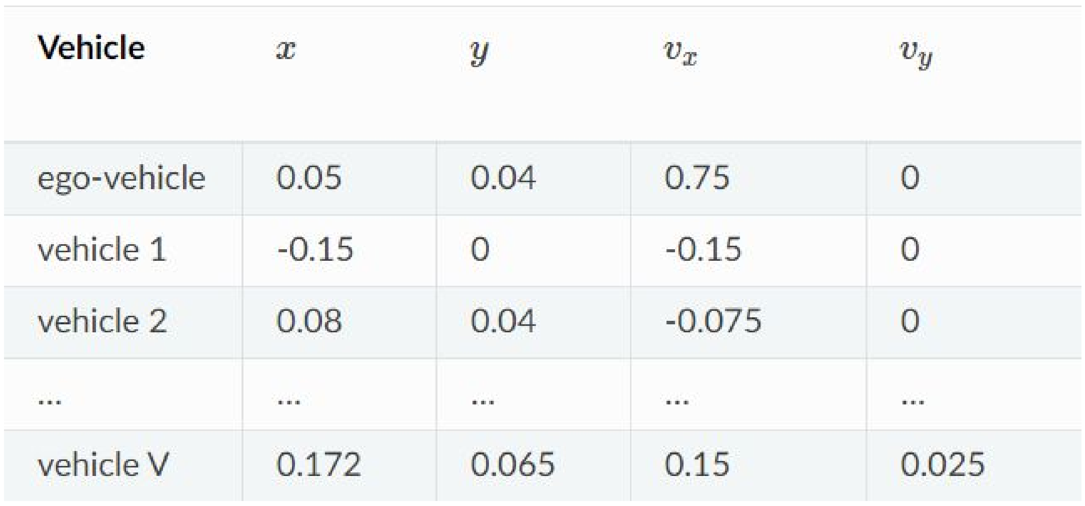


Figure 2: Environment

3.1. State Observations from the environment:

3.1.1. Kinematic Observation:

The *KinematicObservation* is an array of shape (V,F) where V is the number of vehicles in the environment and F is the number of features (for every vehicle). The agent vehicle is referred to as the ***ego-vehicle***. The ego-vehicle is always described as the first element of the *KinematicObservation* array. For every step of the environment, we get the updated values of all the vehicles and their features. For example, if we initialize the vehicles' features to be a set of tuple ***(x, y, vx, vy)*** where x is the position of the vehicle on the x-axis, y is the position of the vehicle on the y-axis, vx is the acceleration of the vehicle along x axis and vy is the acceleration of the vehicle along the y axis, an instance of this *KinematicObservation* would look something like this:



Vehicle	x	y	v_x	v_y
ego-vehicle	0.05	0.04	0.75	0
vehicle 1	-0.15	0	-0.15	0
vehicle 2	0.08	0.04	-0.075	0
...	...	...	...	...
vehicle V	0.172	0.065	0.15	0.025

Table 1: Description of Kinematics Observation Type

3.1.2. Grayscale Image:

The *GrayscaleImageObservation* is an image of shape (W,H) where W is the width and H is the height that is configured while creating the environment. For example, 4 frames stacked together on a grayscale image would look something like this:

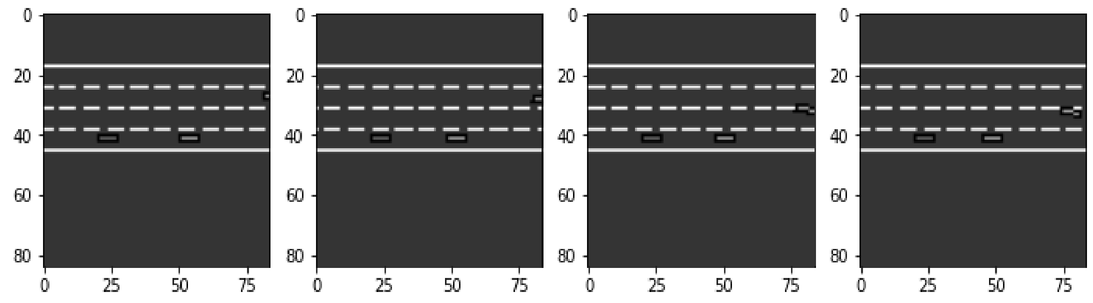
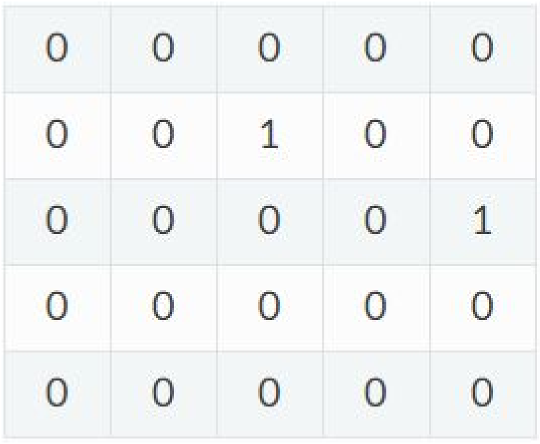


Figure 3: 4-stacked Greyscale Observations

3.1.3. Occupancy Grid:

The *OccupancyGridObservation* is a tensor of shape (W,H,F) that records the observations for all the configured features around the ego-vehicle where W is the width, H is the height and F is the number of features as configured for the environment. Each cell is described by the value of the feature and arrays for all features of a particular cell are stacked upon each other.

For example, presence feature: one vehicle is close to the north, and one is farther to the east.



0	0	0	0	0
0	0	1	0	0
0	0	0	0	1
0	0	0	0	0
0	0	0	0	0

Figure 4: Representation of Occupancy Grid

3.1.4. Time to collision:

The *TimeToCollisionObservation* is a tensor of shape (V,L,H) where V is the number of vehicles, L is the number of lanes and H is the one-hot encodings over discretized time values with 1s steps.

3.2. Actions:

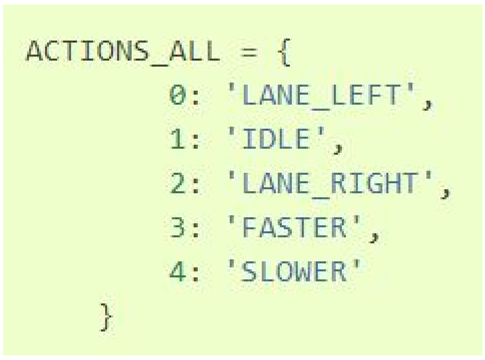
Just like observations, there are choices for the kind of actions the agent can take on the environment. This type of actions that the agent can take can be configured in the environment. The types of actions the agent can take are as follows:

3.2.1. Continuous Actions:

The *ContinuousAction* type allows the agent to modify the ego-vehicle's acceleration and steering angle which are defined as the kinematics of the vehicle. By altering these two the agent can change lanes, accelerate or decelerate and drive along the road.

3.2.2. Discrete Meta Actions:

The *DiscreteMetaAction* type allows the agent to have speed and steering controls so that the ego-vehicle can automatically follow the road at a desired velocity. The action space under *DiscreteMetaAction* is defined as:



```
ACTIONS_ALL = {
        0: 'LANE_LEFT',
        1: 'IDLE',
        2: 'LANE_RIGHT',
        3: 'FASTER',
        4: 'SLOWER'
    }
```

Figure 5: Five actions under DiscreteMeta Actions type

3.2.3. Manual Control:

Under *ManualControlAction* type, the ego-vehicle is controlled by directional arrow keys and the agent can be controlled manually by adding a set of actions to the action space.

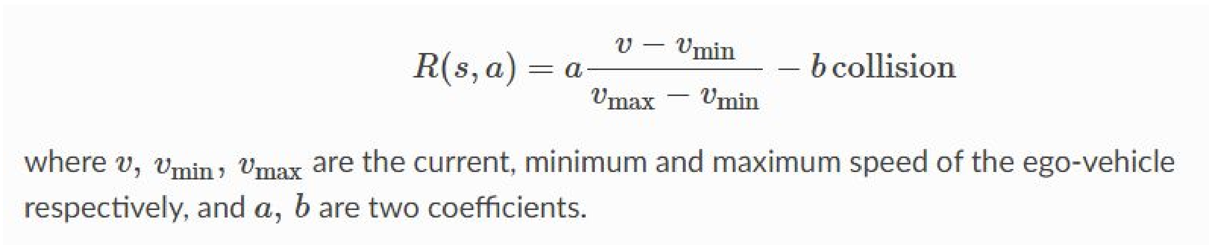
3.3. Rewards:

Our main aim is to make the agent learn to drive safely by adhering to two basic rules:

a. The vehicle should move as fast as possible
b. The vehicle should avoid collisions

Thus, the reward function is evaluated by taking these two factors into account.

The reward function is composed of a velocity term and a collision factor:



$$R(s,a) = a\frac{v - v_{\min}}{v_{\max} - v_{\min}} - b\,\text{collision}$$

where v, $v_{\min}$, $v_{\max}$ are the current, minimum and maximum speed of the ego-vehicle respectively, and a, b are two coefficients.

The weight of the collision penalty can be configured through the collision_penalty parameter.

4. Models Implemented

The configuration of the environment was kept constant across all methods to fairly compare the approaches and evaluate the best agent that can drive on this highway environment. All the environments were configured with the following values:

Parameter	Description	Value
policy_frequency	Controls the frequency at which decisions are taken, and thus that at which transitions (s,a,s') are observed	10
simulation_frequency	Controls the frequency at which the world is refreshed. Increasing it will increase the accuracy of the simulation, but also the computation time	10
vehicles_count	Number of other vehicles observed on a single episode	20
duration	Number of steps for a single episode	3000
num_lanes	Number of lanes on the highway	4

Table 2: Parameters used for configuring the environment

4.1. Double Dueling DQN (DDDQN):

4.1.1 Set the Observation type to *GreyscaleObservation* and action type to *DiscreteMetaAction* and implemented Double Dueling DQN on this environment configuration.

Hyperparameters used for this experiment are as follows:

Hyperparameter	**Value**
Learning Rate	0.0001
Batch Size	32
Discount Factor	0.99
Memory Size	100000

Initial Memory Fill	10000
Replace Target Network	10000
Epsilon Decay	0.000001
Minimum Epsilon	0.1

Table 3: Hyperparameters used for DDDQN

The agent trained using this method seems to have learned the best policy when compared to all other methods in this project. It has learned to overtake efficiently, even when there are multiple cars ahead in close proximity. It has also learned other desirable behaviours like slowing down at the right times to avoid collision and speeding up when there is free space ahead of it. All these behaviours lead to an agent with human-like decision making skills on the highway.

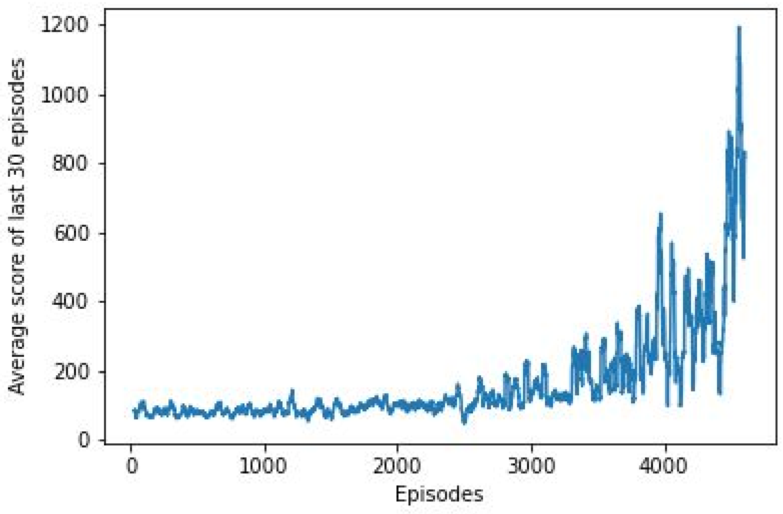


Figure 6: Plot of reward vs number of episodes for DDDQN

4.1.2 Set the Observation type to *TimeToCollissionObservation* and action type to *DiscreteMetaAction* and implemented Double Dueling DQN on this environment configuration.

Hyperparameters used for this experiment are as follows:

Hyperparameter	Value
Learning Rate	0.0001
Batch Size	32

Discount Factor	0.99
Memory Size	70000
Initial Memory Fill	10000
Replace Target Network	10000
Epsilon Decay	0.000001
Minimum Epsilon	0.01

Table 4: Hyperparameters used for DDDQN

The observation was flattened and fed into a feed forward neural network with 3 fully connected layers and ReLU activation function and trained on MSE Loss with RMS prop as the optimizer. The agent trained using this method has learned to slow down when it sees a vehicle in front of itself on the same lane. But, it does not change lanes and pass the vehicle as that is not explored yet by the agent during training. Yet, it drives safely by not colliding with other vehicles and slowing down at the right moment.

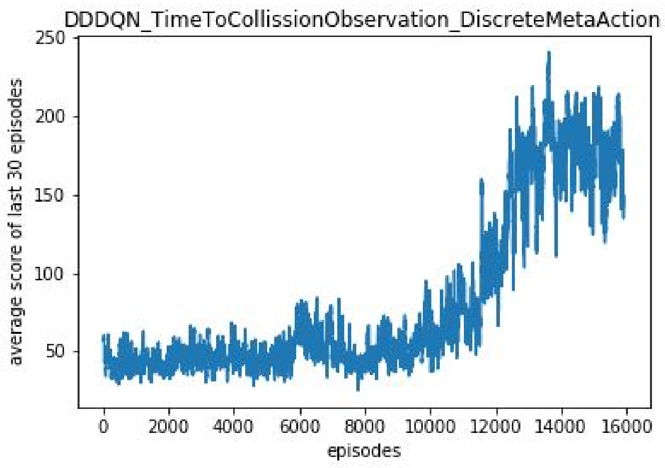
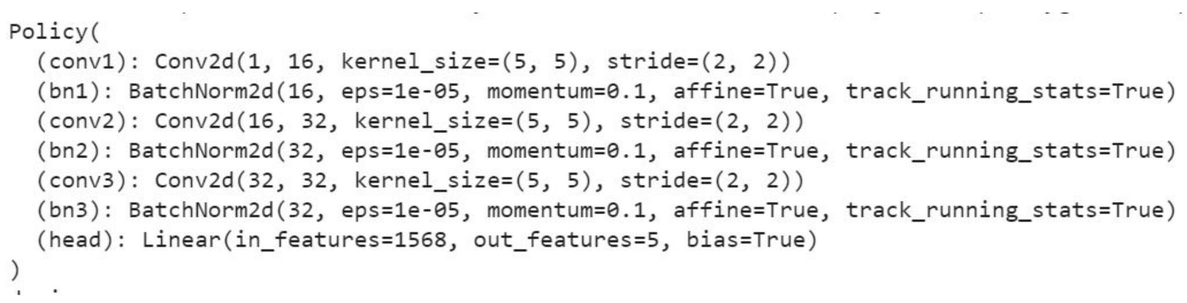


Figure 7: Plot of reward vs number of episodes for DDDQN

4.2. Policy Gradient Reinforcement Learning:

4.2.1 Vanilla Policy Gradient (VPG): As observed in the paper, vanilla policy gradient worked well so we have modeled policy gradient with observation type as *GreyscaleObservation* and action type to *DiscreteMetaAction* by modelling the reward to be a linear value in [0,0.4] for the speed range of the agent and a collision reward of -1 at every step. In the model we have added CNN layers with conv2d layers. Following is the screenshot of model and hyperparameters used



```
Policy(
  (conv1): Conv2d(1, 16, kernel_size=(5, 5), stride=(2, 2))
  (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv2): Conv2d(16, 32, kernel_size=(5, 5), stride=(2, 2))
  (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (conv3): Conv2d(32, 32, kernel_size=(5, 5), stride=(2, 2))
  (bn3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (head): Linear(in_features=1568, out_features=5, bias=True)
)
```

Hyperparameter	Value
Learning Rate	0.00001
Discount Factor	0.99

Table 5: Hyperparameters used for VPG

Average reward over 30 episodes was around 75. Model could not learn after running for 12k episodes. Agent is only able to change lanes but not able to avoid collision.

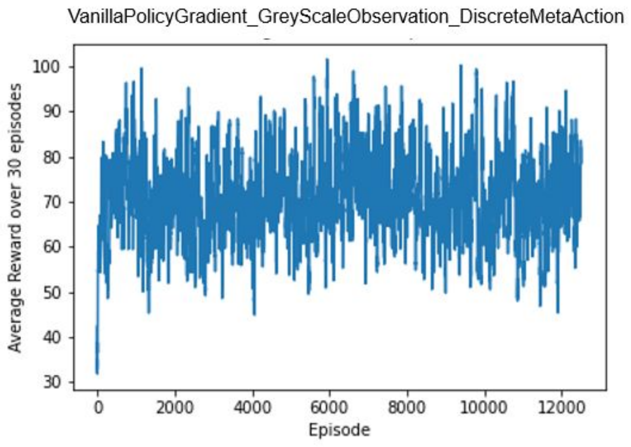


Figure 8: Plot of reward vs episodes for VPG

4.2.2 Deep Deterministic Policy Gradient (DDPG): As DDPG's work best under continuous action spaces, we implemented a DDPG network on *KinematicsObservation* and *ContinuousActions*. The agent was trained using the same configuration for actor and critic networks with two fully connected layers and ReLU activation function. The agent learned to drive on the road and move forward accordingly, but due to lack of training time, it could not learn to drive without crashes and fluctuating its steering angle, but will learn to keep those parameters steady upon more training. The result of the model is as follows :

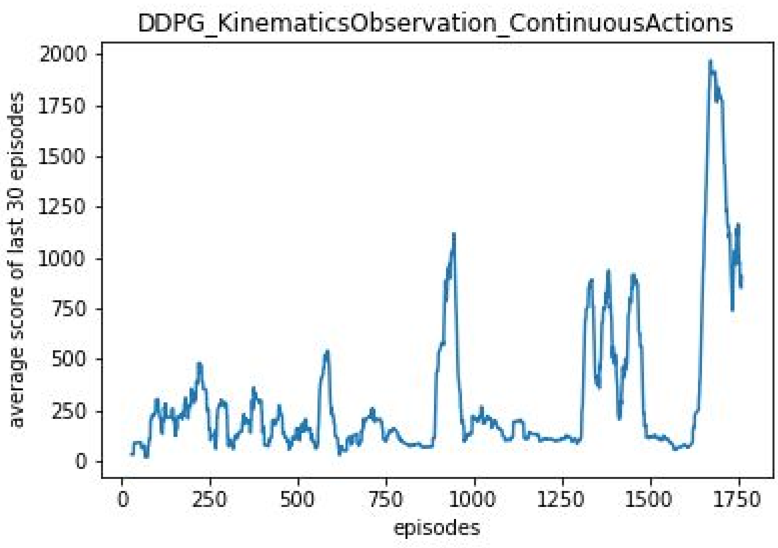


Figure 9: *Plot of reward vs episodes for DDPG*

4.3. Advantage Actor Critic (A2C) :

4.3.1. Advantage Actor Critic with Kinematics and DiscreteMetaActions :

A2C is a synchronous and deterministic policy method. It has two deep neural networks training in parallel, one for determining the actions and other for the state value.

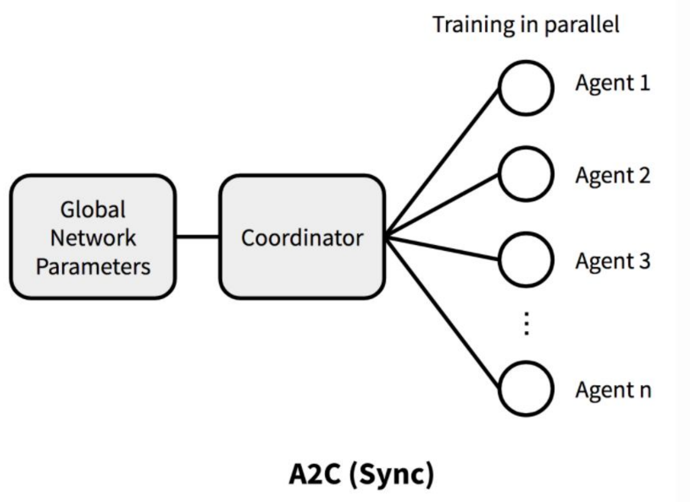


Figure 10: A2C architecture

Since Kinematic observations are easy to train, the model training was much faster than *GreyscaleObservation.* MLP was used for the Deep neural network with two linear layers of 128 and 64. The hyperparameters used are as follows:

Hyperparameter	Value
Learning Rate	0.0001

Discount Factor	0.99

Table 6: Hyperparameters for A2C

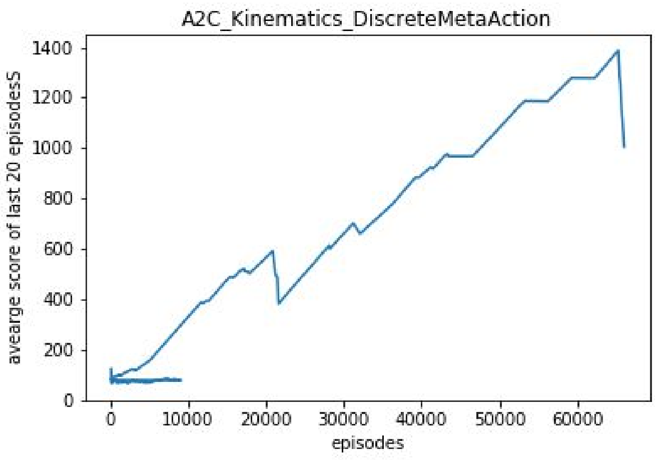


Figure 11: Plot of reward vs episodes for A2C

5. Conclusion:

This project explores the *highway* environment in various state observations and action spaces to determine the best possible way for the environment to be observed for the development of autonomous vehicles. However, this project is limited only to numerical state spaces or aerial views and does not take sensors or camera data sources into account. Various modern Reinforcement Learning methods have been implemented on combinations of state observations and action spaces and their results are as below:

State Observation	Action Space	Algorithm	Average Reward
GreyscaleImage	Discrete	DDDQN	926.13
GreyscaleImage	Discrete	VPG	76.34
Kinematics	Discrete	A2C	676
Kinematics	Continuous	DDPG	1057.53
TimeToCollission	Discrete	DDDQN	223.63

Table 7: Results

Even though the rewards observed are high for the above agents, the test runs do not meet all the requirements for few agents. The best driving agent was the one with *GreyscaleImage and DiscreteMeta* actions using DDDQN as it performed both - crash free driving and passing slower vehicles along the episode's run.

Agents trained on *Kinematics and DiscreteMeta* actions using A2C and *TimeToCollission and DiscreteMeta* actions using DDDQN learnt only one of the two safe driving policies - crash free driving. These agents, despite driving on the highway without crashes, only slowed down when another vehicle was observed in the same lane in front of the ego-vehicle and did not pass. These agents followed the vehicle in front and completed the episode at a slower speed by just learning not to crash. These agents need more training to learn to pass other vehicles and also maintain speed to complete the episode.

The only agent with *Continuous* action space trained on Deep Deterministic Policy Gradient Reinforcement Learning had to control two parameters throughout the episode - acceleration velocity and steering angle. This agent could go off-road with its manual steering controls and was the toughest to train. The agent learnt to drive on the road and move forward along the highway and pass other vehicles by changing lanes, but needs more training to drive crash free and hold a steady steering angle while passing.

6. Future Work:

After training these agents a bit more, we will extend these to other environments such as *merge*, *roundabout* and *parking* to build a complete autonomous driving agent. All state observations and action types can be explored as there is a lot of scope by altering the reward functions to enable the agent explore faster and learn better in fewer episodes.

Once the agents learn how to drive in every environment, our aim will be to train one agent that can drive in any environment as it keeps transforming to encounter real world scenarios, though it possesses as a complex problem. Research in this field is limited and can be carried out with new methods and approaches in the future.

7. References :

- Approximate Robust Control of Uncertain Dynamical Systems (Dec 2018)
- Interval Prediction for Continuous-Time Systems with Parametric Uncertainties (Apr 2019)

- Practical Open-Loop Optimistic Planning (Apr 2019)
- α^α-Rank: Practically Scaling α-Rank through Stochastic Optimisation (Sep 2019)
- Social Attention for Autonomous Decision-Making in Dense Traffic (Nov 2019)
- Budgeted Reinforcement Learning in Continuous State Space (Dec 2019)
- Multi-View Reinforcement Learning (Dec 2019)
- Reinforcement learning for Dialogue Systems optimization with user adaptation (Dec 2019)
- Distributional Soft Actor Critic for Risk Sensitive Learning (Apr 2020)
- Task-Agnostic Online Reinforcement Learning with an Infinite Mixture of Gaussian Processes (Jun 2020)
- Beyond Prioritized Replay: Sampling States in Model-Based RL via Simulated Priorities (Jul 2020)
- Robust-Adaptive Interval Predictive Control for Linear Uncertain Systems (Jul 2020)
- SMART: Simultaneous Multi-Agent Recurrent Trajectory Prediction (Jul 2020)