

Learning Adaptive Driving Behavior Using Recurrent Deterministic Policy Gradients

Kaustubh Mani¹, Meha Kaushik², Nirvan Singhania¹ and K. Madhava Krishna¹

Abstract—In this work, we propose adaptive driving behaviors for simulated cars using continuous control deep reinforcement learning. Deep Deterministic Policy Gradient(DDPG) is known to give smooth driving maneuvers in simulated environments. Unfortunately, simple feedforward networks, lack the capability to contain temporal information, hence we have used its Recurrent variant called Recurrent Deterministic Policy Gradients. Our trained agent adapts itself to the velocity of the traffic. It is capable of slowing down in the presence of dense traffic, to prevent collisions as well to speed up and change lanes in order to overtake when the traffic is sparse. The reasons for the above behavior, as well as, our main contributions are: 1. Application of Recurrent Deterministic Policy Gradients. 2. Novel reward function formulation. 3. Modified Replay Buffer called Near and Far Replay Buffers, wherein we maintain two replay buffers and sample equally from both of them.

I. INTRODUCTION

Over the last few decades, Artificial Intelligence has shown promising advances in various fields: medicine, economics, sciences and transportation. One field in which it has proven to be exciting as well as challenging to the research community is the Autonomous vehicle research. Safety holds the highest priority in Autonomous vehicle research, hence it has attracted the attention of many researchers over the past many years.

A very popular work is [1], wherein the authors have detailed the various high risk design choices(side effects, reward hacking, scalable supervision and distributional shift) for machine learning agents, particularly reinforcement learning. We understand from this literature that the choice of reward function and exploration policy plays a vital role in the quality of learned policy. Poor choice in either of the two can lead to potential risks.

Authors in [2] have proposed a multi agent safe reinforcement learning framework for autonomous driving. Here, the authors have solved for long term driving strategies by minimizing the estimation of gradient ascent and by introducing a concept of "Option Graph" which effectively reduces the variance in gradient estimation. The main contribution is to develop a method that ensures functional safety. Last decade has witnessed huge efforts in the development of autonomous driving cars, driven by the same authors in [3] have formally introduced two parameters(standardization of safety assurance and scalability) to judge the various competitors. Since safety is the foremost important consideration for any machine, its maximal guarantee is utmost for the acceptance

of autonomous vehicles. Second, even if one autonomous vehicle is developed efficiently, it should be reproducible. Safe driving has always been a topic of deep interest in the community. Authors in [4] have also proposed a safe driving method using Multiple Criteria Decision Making for urban traffic conditions.

Deep Deterministic Policy Gradients[5] have been able to generate various driving behaviors in simulated environments, [6], [7], [8]. A major limitation in these behaviors is the high number of collisions with other vehicles. Since the agent did not have information regarding other vehicle's velocities, avoiding collisions completely was difficult. To encode the temporal information in the neural network, we have used a variant of a standard continuous control deep reinforcement learning method Deep Deterministic Policy Gradient, called Recurrent Deterministic Policy Gradient (RDPG). RDPG is introduced in [9], instead of the feed-forward neural networks, recurrent networks are used for actor and critic networks. The use of recurrent networks is intuitive, because driving decisions will be made better with the knowledge of recent history of the environment. We observe that RDPG takes lesser time to train than DDPG in exact same other conditions. Our main contribution is the reward function for adaptive behavior. The highlight of the same is the fact that it can adjust to the traffic conditions around it. It is able to display features like slowing down in congested spaces, changing lanes and overtaking when permissible. The most important feature of the reward function is to prioritize safety by avoiding collisions as much as possible. We compare the proposed reward function with reward functions shown in [7] and [8] and show that the number of collisions decreases hugely for our case. A direct application of this behavior is the ability to generate custom traffic in simulators, which henceforth can be used to create detailed unstructured environments for Autonomous Vehicle training and validating. Another contribution we claim is the use of a modified Replay Buffer. We introduce Near and Far Replay Buffers, wherein samples of two different attributes are stored and while training we sample equally from both of them so as to show both types of examples to our agent equally. Lastly, we compare our results for DDPG and RDPG and show RDPG is faster and produces better quality behaviors.

II. MOTIVATION

A. Defining Adaptive Behavior

Adaptiveness is defined as quickly changing your own attributes or parameters to suit the external conditions. For

¹Robotics Research Center, IIIT-Hyderabad

²Microsoft Canada Development Center

intelligent vehicles, adaptiveness is characterized when the vehicle adjusts its velocity, steering angle according to the environment. The most important contributing factors in the environment are velocities and relative positions of the cars in the surrounding. Typically, if the traffic cars are moving at slow speeds, the agent should also slow down, if they are moving at high speeds then it should speed up to maintain its relative position with respect to other vehicles. If the space on road allows, then the agent should be able to overtake the other vehicles, importantly, the agent should avoid collisions at every step. Hence, adaptive behavior can be summed as greedy(with respect to velocity) driving but with safety in mind.

B. Recurrent Deterministic Policy Gradients

Based on Deterministic Policy Gradients[10], DDPG is one of the most successful continuous control frameworks using deep reinforcement learning. It has shown promising results for control in autonomous vehicles, both in simulators [7],[8], [11] and on self driving hardware [12]. It is an off-policy actor critic[13] based neural network architecture.

The actor network is updated according to:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s)|_{s=s_i} \quad (1)$$

where J is the performance objective which represents the agent's goal to maximize the cumulative discounted reward from the start state. N is the batch-size, θ^Q are the critic network parameters and θ^μ are the actor network parameters. $Q_T(s_{i+1}, \mu_T(s_{i+1}))$ is the target Q value for the state-action pair $(s_{i+1}, \mu_T(s_{i+1}))$ where $\mu_T(s_{i+1})$ is obtained from the target actor network, $Q(s_i, a_i)$ is the Q value from the learned network. Target actor and critic networks are clones of actor and critic networks but are kept fixed for a given number of episodes. They prevent the corresponding network from tailing itself. The target network updates can take place either by directly copying the weights or by using soft update. The equation for soft update is given by:

$$\begin{aligned} \theta^{Q_T} &\leftarrow \tau \theta^Q + (1 - \tau) \theta^{Q_T} \\ \theta^{\mu_T} &\leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu_T} \end{aligned} \quad (2)$$

where θ^μ & θ^Q are the network parameters for the actor and critic networks respectively, θ^{μ_T} & θ^{Q_T} are their corresponding target network parameters and $\tau \ll 1$, is the learning rate.

The updates of the critic network is given by:

$$\begin{aligned} L &= \frac{1}{N} \sum_i (y_i - Q(s_i, a_i))^2 \\ y_i &= (r_i + \gamma Q_T(s_{i+1}, \mu_T(s_{i+1}))) \end{aligned} \quad (3)$$

where r_i is the reward at the i^{th} timestep, y_i is the target value at the i^{th} timestep, N is the batch-size and γ is the discount factor. The rest of the terms have the same meaning as those in Eq. 1.

Another important concept that shaped Deep Reinforcement Learning was that of Replay Buffers. The samples fed

to the RL algorithm are derived from consecutive steps of an episode. Consecutive samples are highly correlated, to avoid training the network with such samples, a replay buffer is used. All the samples are stored in the buffer and during training, the samples are randomly picked up from the buffer.

In real world situations, partially observable control problems are prominently existent. Autonomous Driving itself falls under it. Occluded objects, unknown intent of other vehicles, sharp curves and blind spots are characteristics of any driving scenario. Recurrent neural networks cannot completely solve the problem of blind spots or occlusions but can give a fair estimate because of the encoded history. Replacing the feed forward networks in the Actor Critic Architecture with recurrent memory based networks gives rise to Recurrent Deterministic Policy Gradients [9]. Mathematically, $Q(s, a)$ is replaced by $Q(h, a)$ and $\mu(s)$ by $\mu(h)$. The policy update is given by:

$$\nabla_{\theta^\mu} J \approx \mathbb{E}_\tau \left[\sum_t \gamma^{-1} \nabla_a Q^\mu(h, a)|_{h=h_t, a=\mu^\theta(h_t)} \nabla_{\theta^\mu} \mu(h)|_{h=h_t} \right] \quad (4)$$

where $\tau = (s_1, o_1, a_1, s_2, o_2, a_2, \dots)$ represents entire trajectories drawn from the trajectory distribution induced by the current policy and h_t is the observation-action history at the t^{th} timestep.

Notice, how the expectation is now calculated for entire trajectory, unlike a single state as in equation 1. Back Propagation for recurrent neural networks takes place in form of BPTT i.e. Back Propagation Through Time. The network is unrolled for number of timesteps, error for each step is accumulated, the network is rolled up again and the weights are updated.

Other concepts like Target Network updates, Replay Buffer, they remain same as in DDPG.

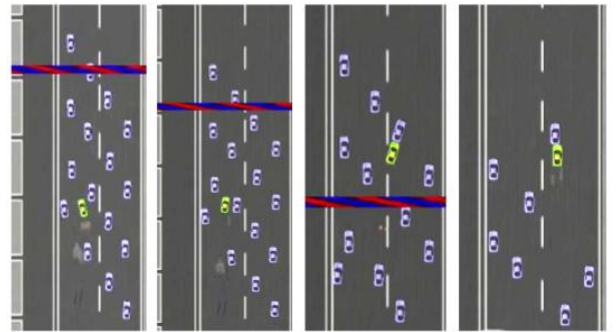


Fig. 1: The above images show overtaking without using adaptive behavior. Observe how the agent crosses the safe minimum distance limit and gets close to the traffic cars.

III. IMPLEMENTATION DETAILS

A. Simulator Details

We have used TORCS [14] which is a widely used platform for research in autonomous driving. The highlight of simulator is its light weighted nature. Unlike the other popular simulators [15] [16], TORCS is not based on unreal

engine, it has a simpler graphics. It does not cater to the entire pipeline of processing the raw sensor values. Instead it offers the processed sensor information. The input to our policy network is a vector of sensor information provided by the simulator which includes ego-vehicle's pose and distance to other vehicles via LiDAR readings.

B. Reward function

Eq.5 and its variants have been the most commonly used reward functions, in similar use cases [7],[8], [11]. This reward function awards the agent for having high velocity component along the desired direction $V_x \cos(\theta)$ and penalizes it for having velocity component perpendicular to the desired direction $V_x \sin(\theta)$. V_x denotes the longitudinal velocity of the car, θ denotes the angle between the car and the track axis. The RL agent trained with this reward function in the presence of dense traffic converges to one of the two policies depending on the collision penalty. If the collision penalty is low, the agent learns a policy which is not safe Fig.1 and greedily tries to maximize its reward by driving at high velocities which leads to many collisions. If the collision penalty is high, the agent learns a policy which is very conservative, and drives at a very slow velocity which results in the agent being nowhere near the traffic, which is not the behavior we seek from our agent.

$$R_{lanekeep} = V_x * (\cos(\theta) - \sin(\theta)) \quad (5)$$

$$R_{adaptive} = \alpha * f(\beta_a, \gamma, minfront_t) * \cos(\theta) \quad (6)$$

$$f(\beta_a, \gamma, x) = \frac{1}{1 + e^{-\beta_a(x-\gamma)}} \quad (7)$$

To this end, we introduce a novel reward function Eq.6 which is designed in order to reward states which are closer to the traffic.

1) *Adaptive Reward*: Our adaptive reward function is designed around a simple intuition, that is, by rewarding states closer to the traffic, the agent (in case of dense traffic) will learn a policy to decelerate if the velocity of the agent is larger than the traffic in front, or accelerate if its velocity is smaller and adapt to the traffic's velocity in order to stay close to the traffic. $minfront_t$ is the distance to the nearest car in the front direction at time t , which is calculated by using the opponent information explained in III-A. α and β_a are hyperparameters which control the scale and shape of the reward function respectively. γ is the margin which controls how close the agent needs to be to the traffic to get high adaptive reward. In Sec. IV, we discuss how this margin can affect the behavior of the agent. One of the nice things about this reward function is that the parameters values can be guessed very intuitively rather than doing a hyperparameter search. Since, α controls the highest adaptive reward the agent can get by staying closer to the traffic it needs to be much larger than the highest achievable lanekeeping reward 5. Eq.7 is simply a scaled and shifted version of the sigmoid function, where β_a is the scale and controls the

smoothness of the boundary between high and low adaptive rewards on either side of the margin γ . γ can be set as the maximum permissible margin from the traffic the agent should maintain.

2) *Overtaking Reward*: To accelerate the training of opportunistic/overtaking behavior, we've defined an additional reward function which is in many ways similar to those used by [7],[8]. One important change from previous versions is that in Eq. 8 we are taking into account the distance from the nearest car defined by $mindist$. The overtaking reward obtained by the agent is lower if the agent overtakes from a close distance to the traffic and higher if the agent overtakes with a bigger margin. This has been done in order to learn a policy in which our agent avoids overtaking in dense traffic situations and overtakes from a safe distance in case of sparse traffic. o_t is the overtaking counter, which represents number of cars overtaken by our agent at time t , η is the maximum possible overtaking reward, $mindist_t$ is calculated by finding the minimum distance in the opponent vector at time t . β_o control the curvature of the overtaking reward function. For high values of β_o , the reward function saturates to η very rapidly, for low β_o , the reward gradually increases to η as the $mindist_t$ is increased.

$$R_{overtake} = (o_t - o_{t-1}) * (\eta(1 - e^{-(\beta_o * mindist_t)})) \quad (8)$$

Our final reward function Eq. 9 is simply the summation of lanekeeping, adaptive and overtaking rewards.

$$R = R_{lanekeep} + R_{adaptive} + R_{overtake} \quad (9)$$

C. Near and Far Replay Buffer

Experience Replay is one of the most important concepts in Reinforcement Learning. Its used to improve the data efficiency of the agent by sampling from past experiences and also the random sampling of batches removes the temporal structure from the data making it independent and identically distributed. The quality of the learned behavior depends entirely upon the experiences populating the replay buffer. Often in continuous control tasks, an agent may encounter experiences belonging to one class more than others. This can lead to large class imbalance in the training batch sampled from the replay buffer.

For learning adaptive driving behaviors, the agent has to learn how to drive in the presence of both dense and sparse traffic. Intuitively, we can understand that the agent will have to exhibit different behaviors in these cases. In the presence of dense traffic the agent should slow down and maintain the velocity of the traffic in order to get high adaptive reward. In case of sparse traffic or when the agent is away from the traffic the total reward Eq. 9 converges to lanekeeping reward $R_{lanekeep}$. In this case the agent must accelerate in order to get high lanekeeping rewards. As the agent slowly learns the adaptive behavior, the replay buffer will be filled by only the states close to the traffic, and the agent will forget how to behave in the case of no traffic or sparse traffic, as each

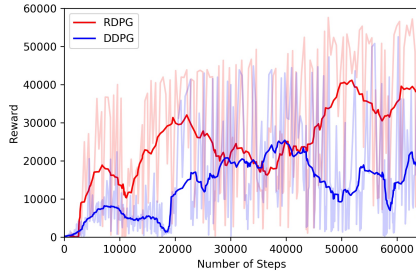


Fig. 2: Comparison of RDPG and DDPG on the task of lane keeping.

batch update will now contain state transitions belonging to dense traffic more than those belonging to sparse traffic.

To show our agent enough examples of both the cases, when it receives high reward with high velocity when it is far from the traffic and when it receives high reward independent of velocity when it is close to the traffic, we sampled data points from two replay buffers: Near replay buffer and far replay buffer. As the name suggests, Near Replay Buffer stores samples wherein our agent is near the traffic vehicles and Far Replay Buffer stores the samples wherein our agent is far from the traffic vehicles. Near and far is defined by the same margin parameter γ explained in Sec. III-B.1. If the $minfront_t$ parameter is smaller than the margin γ , the state transition is added to the Near buffer. If $minfront_t$ is larger or equal to γ , the state transition is added to the Far buffer. While sampling for batch update, equal number of samples are picked from both of the replay buffers. As we will later see in Section IV, the addition of Near and Far buffers leads to gain in sample efficiency resulting in faster training and better generalization during test time.^{1 2}

IV. RESULTS

A. Experimental Setup

We have performed extensive experiments to evaluate the performance of our reward function and compare the two algorithms DDPG and RDPG and their variants. We have trained our agent with 16 traffic cars with having velocities sampled from a uniform distribution between 25km/hr and 35km/hr. Traffic is divided into two blocks of 8 cars each, the velocity of cars in a block is kept the same. The positions of traffic cars are also chosen randomly from a set of possible formations to simulate dense and sparse traffic situations. During training, the traffic cars start in a dense formation, and then randomly change their positions. Positions and velocity of traffic cars are randomly chosen after every 50 timesteps. However, during testing, in order to measure the robustness and generalizability of our agent, velocities of traffic agents are sampled from a uniform distribution between 5km/hr and 105km/hr and the position of traffic is also randomly chosen from a larger set of formations every 20 timesteps. To do a fair comparison, we keep a random seed for all of our testing simulations in order to generate the same sequence

¹Link to the results: <https://goo.gl/b8wyt1>

²Link to the code: <https://goo.gl/sqMZEh>

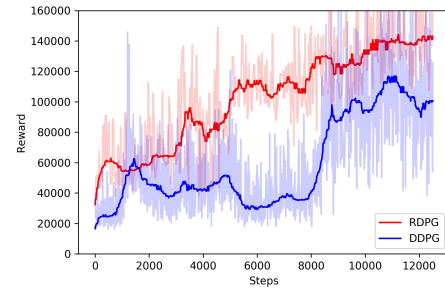


Fig. 3: Comparison of DDPG and RDPG on the task of adaptive driving.

of random values. We use the following metrics to evaluate the performance of our agents:

1) *Collisions*: In Table I, the row Collisions represent the total number of collisions incurred by the agent over a period of 1000 episodes each of 1000 timesteps. In this case, episodes are immediately terminated if the agent collides with the traffic. In Table II, we are comparing the percentage of timesteps when there is a collision between our agent and any of the traffic cars over the period of 1000 episodes each of 100 timesteps.

2) *Min Front*: In Table I, Min Front is calculated by averaging the $minfront$ parameter explained in Sec. III-B.1 over the period of 1000 episodes, for the first 100 timesteps when the traffic is in dense mode. Min Front value represents how close the agent is able to safely drive in the presence of dense traffic. As we are testing with varying traffic velocities different from those at the training time. A low value of Min Front tells us that the agent is able to stay close to the traffic which means the agent is able to better adapt to the velocity of the traffic and thus shows better generalizability.

3) *Cars Overtaken*: In Table I Cars overtaken is calculated by finding the average of overtaking counter o_t explained in Sec. III-B.2 obtained at the end of each episode over a period of 1000 episodes. This parameter comments on the overtaking capabilities of the agent. The agent which is able to overtake more cars on average adapts better to the diverse traffic scenarios.

During training, we are adding Ornstein-Uhlenbeck [17] noise to each of three actions (Steering, Acceleration, Brake) for exploration using ϵ -greedy method. For lane keeping, we train our agent for 100000 timesteps and for another 200000 timesteps to learn adaptive and overtaking behaviors respectively. Fig. 2 and 5 shows the training plots for lane keeping and adaptive behaviors respectively. The agent was trained with α equal to 1000, β_a equal to -0.5, γ equal to 30m, β_o equal to 0.2, η equal to 10000, learning rate for actor being 0.0001 and 0.001 for critic, buffer size of 200000 for single buffer and near and far buffers each of size 100000 in case of two buffers, batch size of 32. The value of τ for target network is set to 0.001. RDPG is trained with every sample in a batch having 10 timesteps, one for the current state transition and 9 past state transitions. Analysis for both Table I and II are done with the agent trained with the hyperparameters mentioned above.

Track Name	Parameter	DDPG-v1	DDPG-v2	RDPG-v1	RDPG-v2
CG1	Collisions	98	91	48	42
	Min Front	56.592	39.452	49.754	35.182
	Cars overtaken	10.452	14.532	10.878	14.978
CG2	Collisions	104	93	57	59
	Min Front	62.821	42.122	57.964	40.151
	Cars overtaken	9.819	12.112	8.882	14.084
CG3	Collisions	87	88	42	31
	Min Front	49.213	36.197	42.991	31.886
	Cars overtaken	11.213	15.128	12.527	15.741
Street1	Collisions	94	96	61	36
	Min Front	55.123	41.125	50.986	44.746
	Cars overtaken	10.972	13.926	12.123	13.984

TABLE I: The above table compares DDPG and RDPG with and without Near Far Replay Buffers. This data has been calculated over the period of 1000 episodes, with each episode having 1000 steps.

Track	No of cars	Lanekeeping		Adaptive	
		DDPG	RDPG	DDPG	RDPG
CG1	4	4.987	2.762	0.0116	0.00812
	9	9.871	4.902	0.0136	0.00791
	16	42.842	19.773	0.0231	0.0113
CG2	4	3.749	1.858	0.0127	0.0104
	9	10.081	6.134	0.0157	0.00926
	16	33.624	17.782	0.0312	0.0203
CG3	4	3.782	1.212	0.00913	0.00428
	9	5.925	1.984	0.0148	0.00948
	16	36.795	12.489	0.0382	0.0185
Street	4	4.852	2.997	0.0104	0.00752
	9	10.328	4.892	0.0144	0.0082
	16	39.792	22.196	0.0331	0.00906

TABLE II: Comparing reward functions in terms of % of colliding timesteps

B. Comparing DDPG and RDPG

We compare the two deterministic policy gradients algorithms on their training time and performance. Fig. 2 uses the reward versus steps plot to compare the two algorithms on the task of lane keeping. It can be observed from the plot that RDPG learns faster than DDPG and also saturates to a higher reward which means that the RDPG agent is able to drive faster while maintaining its lane in comparison to the DDPG agent. Fig. 3 compares the two algorithms on the task of adaptive driving and overtaking. The plot shows that RDPG is more sample efficient and learns faster in comparison to DDPG. Also, From Table I, we can see that RDPG performs better than DDPG in both scenarios i.e with or without Near and Far buffers. Because of the recurrent connections, it can encode temporal information about the traffic i.e. (the relative velocity and acceleration of the surrounding traffic) which helps RDPG to learn a better policy. The resultant policy is better at avoiding collisions and generalizes better than DDPG to the traffic not seen during training time. Table II shows that the RDPG agent performs far better than DDPG agent even in the case of lanekeeping reward. Qualitative results show that RDPG with lanekeeping reward learns defensive behavior which means that the agent brakes instantaneously to avoid collisions once it is very close to the traffic. This is not the desired behavior as it leads to oscillations in the movement of the agent.

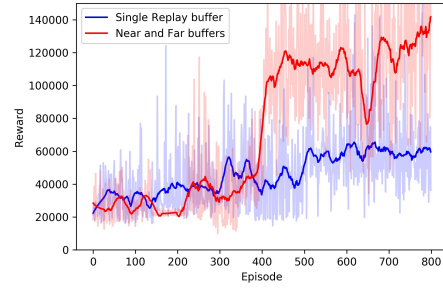


Fig. 4: Comparison of RDPG with a single Replay Buffer and RDPG with our Near and Far Replay Buffer on the task of adaptive driving.

C. Results indicative of our Reward Function

The objective of this work has been to learn an agent which can navigate safely in dense traffic environments by adapting itself to the velocity of the traffic, and show opportunistic/overtaking behaviors once there is enough space for the agent to overtake safely. Figure 1 displays the behavior of the agent trained using the old reward function(i.e. $R_{lanekeep}$). As can be seen from the figure that even in dense traffic situations, the agent tries to unsafely overtake traffic cars and eventually leading to a collision. Analysis shows that although the agent is able to navigate through sparse traffic without many collisions it always ends up colliding with the traffic in case of dense traffic scenarios.

Figure 5 shows the results of the adaptive behavior learned using the proposed reward function. We can observe that in our approach the agent always maintains a safe distance from the traffic by adapting to the traffic's velocity and it's also able to overtake other cars within safety limits. The agent drives less aggressively when it is close to the traffic and display behaviors similar to human drivers in the same situation.

Table II gives a quantitative comparison between the two reward functions. It can be seen that our proposed reward function drastically decreases the number of collisions on all tracks. We can also see that when the number of traffic cars is low, the collision rate is low for both reward functions but as we increase the number of traffic cars and make the traffic more and more denser, the collision rate for lane-keeping reward rises dramatically. In comparison, the collision rate for the agent trained with our proposed reward function increases only slightly as the traffic gets denser.

D. Importance of Near and Far Replay Buffer

As discussed in Sec. III-C, the addition of Near and Far Replay Buffer can lead to better sample efficiency and generalization. We can see in figure 4, the traditional use of replay buffer leads to our trained agent converging to a sub-optimal policy, on the other hand, the use of Near and Far Replay Buffers leads to a high value of reward after 400 episodes of training. We can infer from it that, near and far replay buffers, lead to a high value of reward and hence facilitated the learning of better policy in less number of episodes than the traditional, single replay buffer. From Table I we can see that the for both DDPG and RDPG with

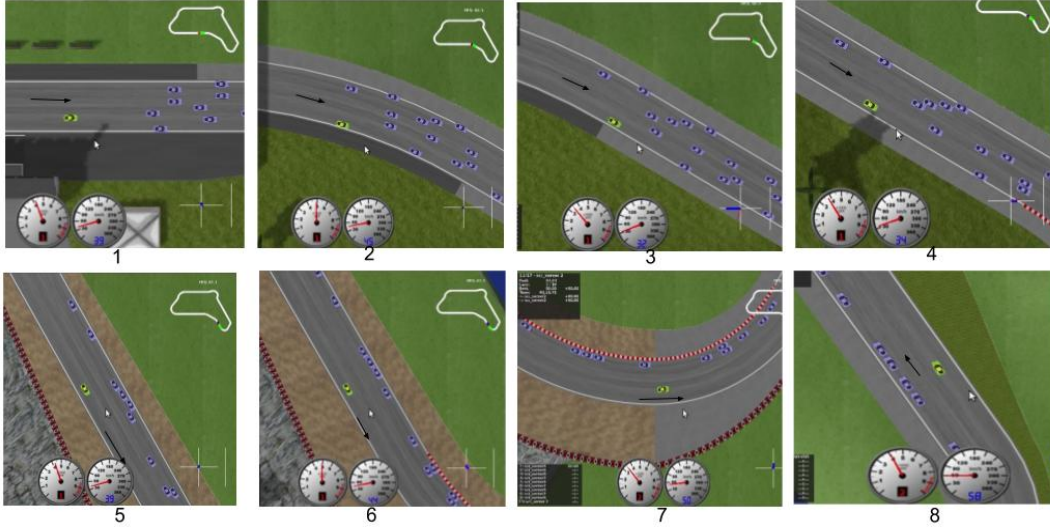


Fig. 5: Figure showing adaptive behavior. Blue cars are the traffic cars and green is the trained agent. The speedometer in the images displays the speed of our agent. The plot shows our agent adapting to the traffic's velocity in case of congesting in traffic(1-4) and then overtaking in sparse traffic situations(4-8).

the addition of Near and Far Replay buffers, the collision rate goes down slightly, but more importantly the average front distance from the traffic significantly decreases indicating that the agent is able to stay more close to the traffic which means that the agents get high adaptive rewards during test time and hence shows more generalization.

E. Effect of Hyperparameters on Optimal Policy

As we mentioned in Sec. III-B.1, the margin parameter γ in our reward function can control the behavior of the agent. Fig. 6 shows that for low margin values the collision is high and also that the minimum front distance maintained by the agent (Min Front) with the traffic is low. As we increase the margin value in our reward function Eq. 6, the number of collisions decreases and the agent maintains increasingly higher distances and becomes safer.

β_o in Eq. 8 controls how quickly the overtaking reward saturates to η as we increase the *mindist* value. If β_o is small, it means that the agent will not get a high overtaking reward if it overtakes by a smaller distance. If β_o is high, it means that the agent can get a high reward even for overtaking by a small distance. Fig. 7 shows that as we increase the value of β_o the average Overtaking distance which is the distance to the nearest car when our agent is overtaking another car decreases which means that the agent is not overtaking safely leading to higher number of collisions. It can also be seen that as we increase the value of β_o , the agent overtakes less safely and collision increases.

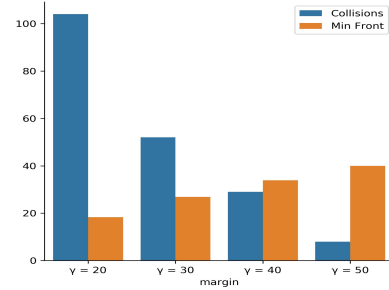


Fig. 6: Analyzing how margin γ in Eq. 6 affects the adaptive driving behavior

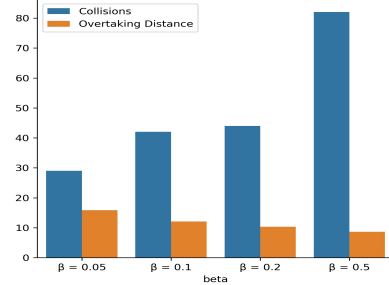


Fig. 7: Analyzing how β_o in Eq. 8 affects the overtaking behavior

V. CONCLUSION

To account for temporal information in Deep Deterministic Policy Gradients, we introduced recurrent networks. We compared the training rate and results produced by both DDPG and RDPG, we observed RDPG learns the optimal behavior at a faster rate. We have also introduced a reward function to learn adaptive behavior. The learned agents are able to maintain lane and undertake overtaking trajectories if possible safely.

REFERENCES

- [1] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.
- [2] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*, 2016.
- [3] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. On a formal model of safe and scalable self-driving cars. *arXiv preprint arXiv:1708.06374*, 2017.
- [4] Andrei Furda and Ljubo Vlacic. Enabling safe autonomous driving in real-world city traffic using multiple criteria decision making. *IEEE Intelligent Transportation Systems Magazine*, 3(1):4–17, 2011.
- [5] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.
- [6] Meha Kaushik, K Madhava Krishna, et al. Parameter sharing reinforcement learning architecture for multi agent driving behaviors. *arXiv preprint arXiv:1811.07214*, 2018.
- [7] Meha Kaushik, Vignesh Prasad, K Madhava Krishna, and Balaraman Ravindran. Overtaking maneuvers in simulated highway driving using deep reinforcement learning. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1885–1890. IEEE, 2018.
- [8] Meha Kaushik and K Madhava Krishna. Learning driving behaviors for automated cars in unstructured environments. In *European Conference on Computer Vision*, pages 583–599. Springer, 2018.
- [9] Nicolas Heess, Jonathan J Hunt, Timothy P Lillicrap, and David Silver. Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*, 2015.
- [10] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.
- [11] Ahmad El Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. End-to-end deep reinforcement learning for lane keeping assist. In *NIPS Workshop on Machine Learning for Intelligent Transportation Systems (MLITS)*, 2016.
- [12] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to drive in a day. *arXiv preprint arXiv:1807.00412*, 2018.
- [13] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.
- [14] Bernhard Wymann, Eric Espié, Christophe Guionneau, Christos Dimitrakakis, Rémi Coulom, and Andrew Sumner. TORCS, The Open Racing Car Simulator. www.torcs.org, 2014.
- [15] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2017.
- [16] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.
- [17] George E Uhlenbeck and Leonard S Ornstein. On the theory of the brownian motion. *Physical review*, 1930.