Overtaking Maneuvers in Simulated Highway Driving using Deep Reinforcement Learning

by

Meha Kaushik, Vignesh Prasad, Madhava Krishna

in

2018 IEEE Intelligent Vehicles Symposium (IV) (IV'18)

Report No: IIIT/TR/2018/-1

Overtaking Maneuvers in Simulated Highway Driving using Deep Reinforcement Learning

Meha Kaushik[1], Vignesh Prasad[2], K Madhava Krishna[1] and Balaraman Ravindran[3]

Abstract— **Most methods that attempt to tackle the problem of Autonomous Driving and overtaking usually try to either directly minimize an objective function or iteratively in a Reinforcement Learning like framework to generate motor actions given a set of inputs. We follow a similar trend but train the agent in a way similar to a curriculum learning approach where the agent is first given an easier problem to solve, followed by a harder problem. We use Deep Deterministic Policy Gradients to learn overtaking maneuvers for a car, in presence of multiple other cars, in a simulated highway scenario. The novelty of our approach lies in the training strategy used where we teach the agent to drive in a manner similar to the way humans learn to drive and the fact that our reward function uses only the raw sensor data at the current time step. This method, which resembles a curriculum learning approach is able to learn smooth maneuvers, largely collision free, wherein the agent overtakes all other cars, independent of the track and number of cars in the scene.**

I. INTRODUCTION

Over the last few decades, the area of autonomous driving has made significant progress owing to the rise of low cost sensors, availability of vast amounts of driving data, and the boom of Learning based methods. Owing to the rise of deep learning, end-to-end methods have got popular in recent years, which try to learn driving decisions directly from sensory inputs. This way the system learns an intermediate representation that can give better results rather than learning an accurate representation itself and then taking decisions.

ALVINN [1] was one of the first works to explore the autonomous driving space using neural networks. Trained on human driver's road data, they use a fully connected neural network to learn a steering wheel direction corresponding to an input image. The authors of [2], [3] approach the task in a similar fashion. In [4], a Recurrent Neural Network is evolved to learn driving behavior in simulation. [5] propose a query efficient Imitation Learning based approach to learn an end-to-end policy for visual autonomous driving in TORCS.

Another approach to the problem is that of Direct Perception [6], [7] where sensory input is mapped to few key affordances, which are used to make a driving decision. Like the above described methods, these require large amounts of training data as well. The need for manual labeling is circumvented by using TORCS[8] for gathering data.

Given that autonomous driving is a situation where an agent makes decisions based on sensory inputs, the problem can be adapted to a Markov Decision Process (MDP) and Reinforcement Learning can be readily applied. The application of Deep RL for such control oriented tasks boomed ever since the super human performance of Deep Q Networks (DQNs) in Atari games [9], [10] and the success of AlphaGo [11]. There are works tackling the problem of autonomous driving using RL [12], [13], [14], [15], however they discretize either the state or action spaces. Other approaches [16], [17], [18] which look at using RL in a continuous setting focus only on lane driving without considering any nearby cars. The authors of [19] apply RL for learning to race in a simulated environment. Few works have also come up showing the use of Inverse RL (IRL) [20] for the task of autonomous driving as well. [21] present a framework to learn costmaps for autonomous driving using IRL directly from sensor data. [22] present an IRL approach in a simple highway driving scenario on a custom simulator.

We follow a similar trend of learning from raw sensory inputs in a simulated environment. We present a novel approach using Deep Deterministic Policy Gradients (DDPGs) [23] to learn continuous actions in a Curriculum Learning [24] like setting by making the agent learn a simpler task first and then moving to learning a more complex task.

Our methodology of training the agent is similar to how humans learn to drive. We are first taught to drive a car straight on an empty road, and keep it from straying away from its path. Once this is done, the driver is made to drive in traffic among other cars to learn to navigate in such scenarios. Similarly, we first train the agent for performing lane keeping. The learned behaviour is then augmented to learn overtaking maneuvers. To the best of our knowledge this is one of the first works to approach the problem in a continuous manner, and one of the first to show the effectiveness of adopting a curriculum learning approach for tackling such problems.

II. BACKGROUND

A. Deep Deterministic Policy Gradients

Deep Deterministic Policy Gradients[23] is a deep version of DPGs inspired from the success of DQNs. They use the two concepts introduced in DQNs along with a third one called Batch Normalization[25].

1) **Replay Buffer**: Transition Tuples, (s_t, a_t, r_t, s_{t+1}), are sampled from the environment as per the exploration policy and stored into a replay buffer. Here s_t, r_t and a_t denote state, reward and action respectively, at timestep, t. We use this representation in the rest

[1]Robotics Research Center, KCIS, IIIT Hyderabad meha.kaushik@research.iiit.ac.in mkrishna@iiit.ac.in
[2]Embedded Systems and Robotics, TCS Innovation Labs Kolkata vignesh.prasad@tcs.com
[2]Dept. of Computer Science, IIT Madras ravi@cse.iitm.ac.in

of the paper, with the omission of the subscript t in some cases for brevity. The correlation between the states of similar trajectories does not allow a stable and convergent learning. Sampling mini-batches of experiences randomly from the buffer solves this issue for both the actor and the critic.

2) **Target Networks**: Instead of directly copying weights, target networks use "soft" updates. A copy of the networks for both the actor and critic are created, denoted by $Q_T(s,a)$ and $\mu_T(s)$ respectively, but their weights are updated by slowly tracking the learned network. This helps in improving the stability of the learning by constraining the weights to change slowly.

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

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 $\tau << 1$, is the learning rate.

3) **Batch Normalization**: Different components of a input to a neural network, usually have different units and scales. This results in slower and inefficient training. Batch Normalization was a solution to resolve this. It normalizes each dimension across the samples in a minibatch to have unit mean and variance. It also maintains a running average of the mean and variance to use for normalization during exploration.

DDPG is an off-policy algorithm, hence the exploration technique is completely independent from the learning policy. It allows us to use simple techniques like adding noise into our actor policy for exploration.

Similar to Q-learning[26], [27], weights of Critic Network are learned using a loss obtained from the Bellman-equation:

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

where r_i is the reward at the i^{th} timestep, $Q_T(s_{i+1}, \mu_T(s_{i+1}))$ 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, N is the batch-size and γ is the discount factor.

The Actor network is updated as given below:

$$\nabla_{\theta^{\mu}} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}}\mu(s)|_{s=s_i} \tag{3}$$

where N is the batch-size, θ^{Q} are the critic network parameters and θ^{μ} are the actor network parameters. The rest of the terms have the same meaning as those in Eq. 2.

B. Curriculum Learning

Curriculum Learning [24] refers to using training strategies to further enhance the performance of a learning agent by presenting it with examples in such a way to help guide the learning process and make it better, rather than randomly presenting examples to the agent to learn from. It not only helps speed up convergence but also provides better local optimum guarantees. It stems from the way humans and animals learn in nature where the complexity of information presented is gradually increased.

III. APPROACH & IMPLEMENTATION DETAILS

Algorithm 1 Overtaking using DDPG

```
$Randomly\ initialize\ Actor\ and\ Critic\ Networks$
$TargetActor \leftarrow ActorNetwork$
$TargetCriticNetwork \leftarrow CriticNetwork$
for $i = 1\ to\ NumEpisodes$ do
    $s \leftarrow ResetTORCS()$
    for $j = 1\ to\ MaxStep$ do
        $action \leftarrow Policy(s)$
        $action \leftarrow action + N$
        $s', r, done \leftarrow Step(action)$
        $Buffer \leftarrow Store(s, a, s', r)$
        if $size(Buffer) > BufferSize$ then
            $batch \leftarrow Sample(Buffer, BufferSize)$
            $Q_T \leftarrow Update(Critic, batch)$
            $Policy \leftarrow Update(Actor, batch, Q_T)$
            $Update\ Target\ networks\ using\ \tau$
        end if
        if $done$ then
            $break$
        end if
    end for
end for
```

We use a modified version of TORCS called Gym-TORCS[28] which has a RL environment incorporated into it. The AI agent car used is "scr_server1". The opponent cars used are of type "scr_server" with velocities randomly initialized from 10-160km/hr. We use a NVIDIA GeForce GTX 1080 GPU for training. We use the DDPG framework to train the actor and critic networks with architectures as shown in Fig. 1 and 2 respectively. The State Vector is a 65 sized array consisting of the following sensor data:

1) Angle between the car and the axis of the track.
2) Track Information: Readings from 19 sensors with a 200m range, present at every $10°$ on the front half of the car. They return the distance to the track edge.
3) Track Position: Distance between the car and the axis of the track, normalized with respect to the track width.
4) SpeedX: As the name suggests, speed of the car along the longitudinal axis of the car.
5) SpeedY: Lateral speed of the car.
6) SpeedZ: Vertical speed of car, indicates bumpiness.
7) Wheel Spin Velocity of each of the 4 wheels.
8) Rotations per minute of the car engine
9) Opponent information: Array of 36 sensor values, each corresponding to the distance of the nearest opponent in the range of 200 meters, located at a difference of $10°$, spanning the complete car.

Further details about each of these sensor readings can be found in [13]. The Action Vector consists of continuous values, the ranges of which are given below:

1) Steer: This represents the steering angle and ranges from -1 (complete right) to 1 (complete left).
2) Brake: This indicates the strength of braking and ranges from 0 (no brake) to 1 (complete braking).
3) Acceleration: This is like the opposite of brake in the sense that it ranges from 0 (no acceleration) to 1 (full acceleration).

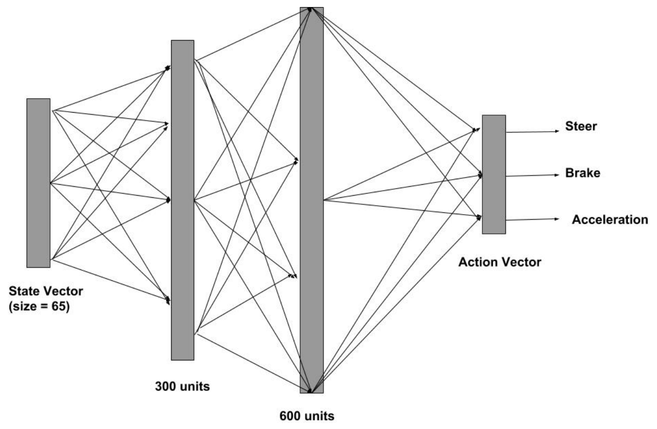

Fig. 1: Neural architecture of the Actor Network

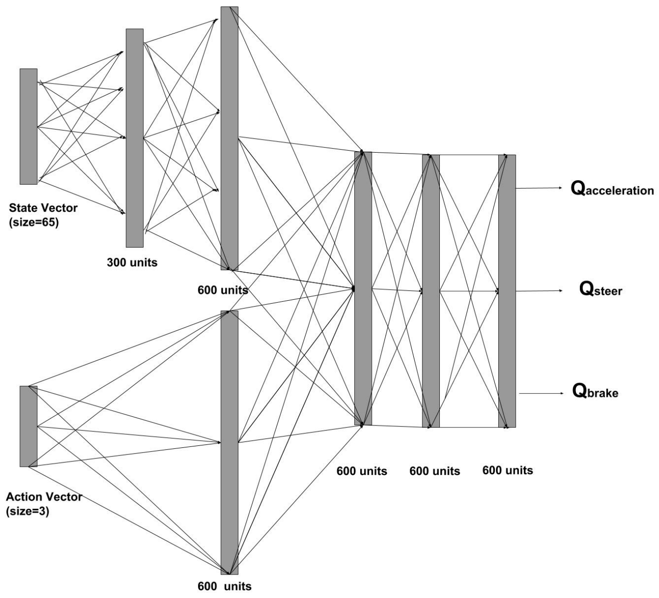

Fig. 2: Neural architecture of the Critic Network

A. Lane Keeping Behaviour

In order to better learn the required behaviour, we have two phases of learning. In the first phase, the agent is made to learn to drive smoothly on the road in a single lane. The reward function for achieving this task, is given below,

$$R_{Lanekeeping} = v_x(cos\theta - sin\theta) - v_x abs(t) \quad (4)$$

where v_x denotes the longitudinal velocity of the car, θ denotes the angle between the car and the track axis. We give a positive reward when the car moves forward along the track axis, given by $v_x cos\theta$, and negative reward when it moves laterally, i.e. perpendicular to the track axis, given by $-v_x sin\theta$. We penalize the car for going off track by giving a negative reward for the amount that the car has moved off the track axis. This amount is denoted by t which is also called the **Track Position** and lies in the range [-1,1]. The probability by which it could go off track is also directly related to the velocity as a higher velocity could cause the car to drift off the track easily as compared to a lower velocity. Hence, we penalize the value $v_x abs(t)$, thereby making the car learn to stay close to the track.

B. Overtaking Behaviour

Once the agent achieved suitable performance for lane keeping, we add more cars in the simulation and augment the reward function to teach the agent to overtake neighbouring cars and navigate in a traffic like scenario. The reward for the second phase of training included the above reward along with a reward for being ahead of other cars so that the overtaking behaviour is favored, as the car would try to get ahead of the others. This reward function is given below.

$$R_{overtaking} = R_{Lanekeeping} + 100 * (n - racePos) \quad (5)$$

Here n denotes total number of cars in a given episode and $racePos$ denotes the position of car in the race, which is obtained from the simulator. If the car is behind other cars, the value of $racePos$ will be higher, thereby decreasing the value of $n - racePos$ giving a lower reward. Once it overtakes a car, the value of $racePos$ decreases, increasing the value of $n - racePos$, giving a higher reward.

Apart from equation 5, explicit rewards were given to handle some special cases:

TABLE I: Extra Rewarding Conditions

Condition	Reward
Collision	-1000
Off track drifting	-1000
No Progress	-500
Overtaking	$R_{overtaking} + 2000$
Overhauling	$R_{overtaking} - 2000$

The explicit conditions of colliding with another car, drifting off the track and not making any progress are necessary, as our reward contains no penalty for them. The extra conditions for Overtaking and Overhauling helped in improving the training rate i.e with these conditions in place the agent learns in less number of episodes. For exploration purpose Ornstein-Uhlenbeck [29] noise is added to all three actions.

In the beginning, for better exploration, noise is kept high. As the agent starts learning the desired behaviour, noise is reduced, for which we have kept a multiplier $\epsilon = 1$ and is reduced by 0.00001 every time step. We train the lane keeping behavior for 2000 episodes and overtaking behavior for 1000, with the learning rate for the actor being 0.0001 and for the critic being 0.001, buffer size was 100000, batch size 32 and γ being 0.99. The value of τ for learning the target network weights is 0.001 in both the cases.

IV. RESULTS

A. Relevant Observations

We analyzed the results produced by various modifications of the current reward function. Our findings are given below:

1) By training the agent on a single track with 4 neighbouring cars, the agent was able to learn for almost all tracks and for number of cars as high as 10.
2) Without the reward for second component of $R_{overtaking}$ in Eq. 5 i.e $100(n - racePos)$, the agent was not able to learn overtaking maneuvers. It drove straight on the lane, colliding with various cars.
3) Without the high reward for overtaking(+2000) and the high penalty for overhauling(-2000), the car was able to learn the overtaking maneuvers after number of episodes as high as 2000.
4) When the high positive and negative rewards mentioned above, were included in the reward function, the car was able to learn by 1000 episodes as seen in Fig 3. This indicates, that high positive reward for a behavior at one time step helped in faster learning.
5) Without pre-training for lane keeping, the car was not able to learn the overtaking maneuvers, up to 4000 episodes. This can be reasoned on the fact that, in the first 1000-1500 episodes the car learned lane keeping behavior, next when it was expected to learn overtaking behavior, the OU noise was reduced greatly and the car was not able to explore sufficiently enough to learn.

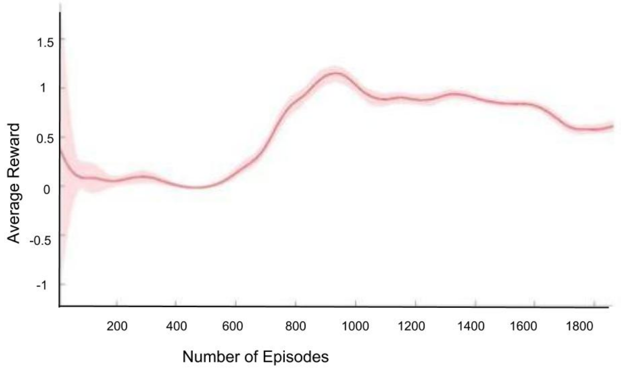

Fig. 3: Graph of Average Reward versus number of episodes during Training. Total time steps = 210000

The highlight of our work is the fact that even by training with 4 neighbouring cars, our car was able to successfully drive and overtake on various tracks with as high as 9 neighbouring cars. We randomly gave the cars speeds in range of 10km/hr to 160km/hr. The trained agent was able to overtake in high speeds as well. Lastly, our car was able to overtake in almost all tracks with minimal collision points. Visually, these can be seen in Fig. 4 and 5. The purple car is our agent and the yellow cars are the neighbouring cars. The trajectory followed by our agent is shown in red, with the black arrow showing the general direction of motion along the road. Although a high negative reward was given on collisions, but collision and overtaking being opposite behaviors, the car did not perform extremely well in terms of completely avoiding collisions.

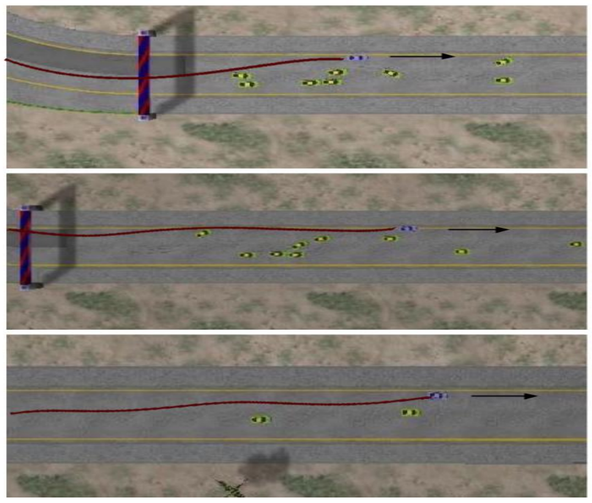

Fig. 4: Results of our approach as viewed from top. The yellow cars are opponent cars and blue is our trained car.

B. Results of DDPG with Curriculum Learning

Table II shows the performance of our agent on various tracks with 4 and 9 neighbouring cars respectively. Out of 9 cars, our agent was able to overtake 7-8 on average. The values in the first column are calculated over 20 episodes, where each episode marks the beginning of all cars starting at 0km/hr from their initial positions. The episode is terminated either when our agent is either out of track or collides disastrously with the walls or other cars, or when it overtakes all other cars. The second column in the table refers to the percentage of timesteps when there was a collision between the agent and another car. In most cases, this value is very low, indicating the collision avoidance nature of this learning.

C. Results of DDPG without Curriculum Learning

In order to get further insights into the effectiveness of our learning, we test how well our approach works without using the Curriculum Learning approach and instead just using the final reward function to learn from scratch. As it can be seen in Fig. 6. Instead of moving around the cars, the agent collides straight into the car, and moves it away from the track. Clearly, it is a very hostile and unsafe behaviour which cannot be counted in overtaking maneuvers.

D. Results of DDPG without Lane Keeping

In order to see the effectiveness of having a lane keeping component to the reward function, we remove the lane

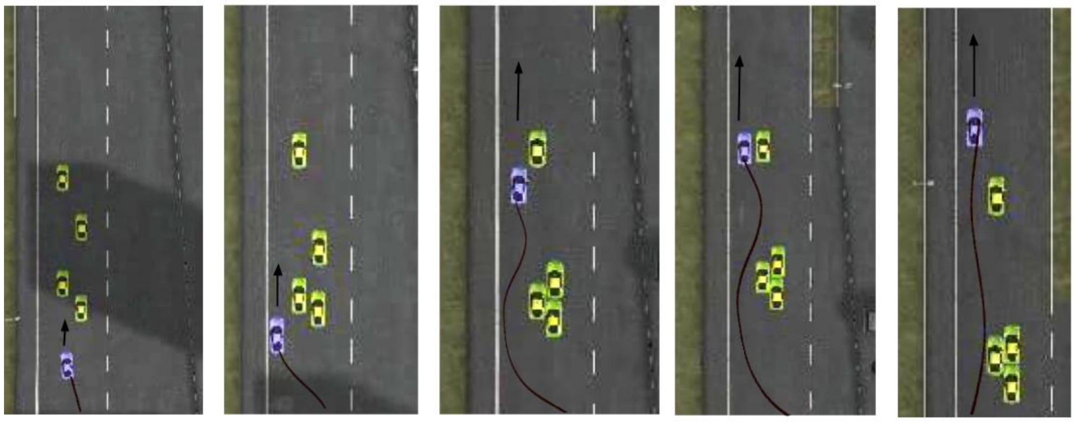
Fig. 5: Results of our approach. The yellow cars are the opponent cars while the purple car is our agent, along with its trajectory shown in red. As it can be seen, our agent successfully learns to overtake cars in front of with without any collisions, or going off the road.

TABLE II: Analysis on Various Tracks with 4 opponent cars and 9 opponent cars in scene

Track Name	Avg no. of cars overtaken		% of colliding timesteps		% of episodes where agent overtook all cars	
	4 cars	9 cars	4 cars	9 cars	4 cars	9 cars
wheel2	3.95	7.55	0.23	0.23	100	50
Forza	4	7.8	25.835	9.64	100	40
CG2	4	8.45	7.01	8.135	95	65
CG3	3.05	6.15	25.64	39.7	30	35
Etrack1	4	8.35	7.08	1.05	100	80
Etrack2	3.55	7.8	26.41	0	65	60
Etrack3	4	6.35	7.99	2.36	100	40
Etrack4	4	8.5	0	7.23	100	70
Etrack6	3.65	7.55	26.13	10.5	90	60
ERoad	4	8.05	3.25	6.9	100	75
Alpine1	4	8.55	17.45	0.67	100	80
Alpine2	3.9	7.95	7.57	0.71	85	50
Olethros	4	7.1	7.3	18.84	100	30
Spring	3.8	7.8	3.87	8.05	95	45
Ruudskogen	3.95	7.65	2.21	12.29	100	40
Street1	3.95	8.55	4.07	6.19	100	80
wheel1	4	8.5	0	10.38	100	50
CG-Speedway1	3.85	7.95	5.62	7.53	95	50

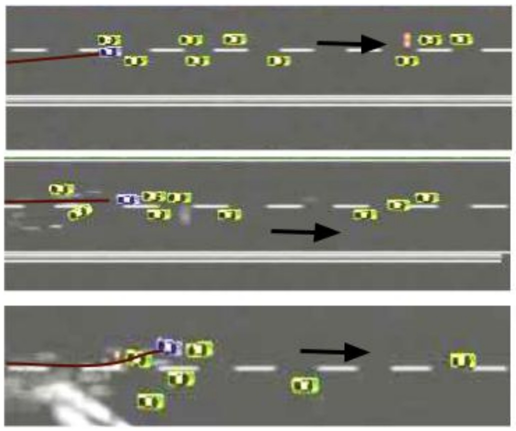
Fig. 6: Agent's Trajectory without using curriculum learning. Neither does the agent properly overtake neighbouring cars, nor does it avoid colliding with them.

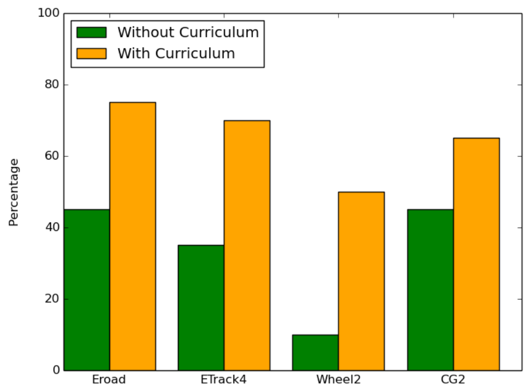

Fig. 7: Percentage of episodes where agent overtook all cars (higher the better).

keeping component and use only the overtaking component of the reward. We do this to see how well the lane keeping helps in learning overtaking behaviour.

Looking further into our results, we observed that, the percentage of episodes where the agent did overtake or at least go ahead of the neighbouring cars reduced drastically (Fig. 7) . Along with this, the percentage of colliding timesteps were much higher i.e. there were much more instances where the agent ended up colliding with neighbouring cars (Fig. 8). Detailed results can be found at: `goo.gl/reKRJ2`

V. CONCLUSION & FUTURE WORK

We present a novel approach for learning overtaking maneuvers in a highway like scenario. Inspired from how humans learn to drive, our approach resembles that of curriculum learning, where we first make the agent learn the simple task of lane keeping followed by adding rewards for learning to overtake. The proposed approach is different from

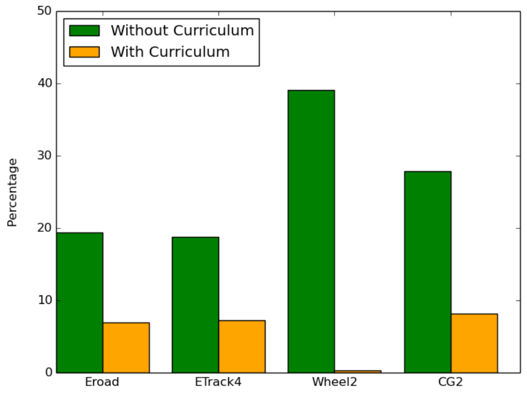

Fig. 8: Percentage of timesteps in which the agent was involved in a collision (lower the better).

most existing methods in the fact that we gradually build up to the task rather than training it for the final task right away. This type of systematic training not only yields more favourable results but does so in lesser time. The learned agent is able to navigate on various tracks while efficiently overtaking neighbouring cars at speeds as high as 160 km/h.

Currently our method uses only information of the current timestep. Incorporating information from the last few timesteps and leveraging the power of Recurrent Neural Networks can help in tackling problems associated with partial observability, occlusion etc. and can help improve the performance of the agent. In city like situations, we would need to augment vision based sensory inputs into the picture as well to account for things like traffic lights, intersections, road signs, footpath detection etc. which would be an interesting problem to solve with our approach. Another aspect is that some of the sensor data, such as track information, track position etc. are not directly available in a real driving scenario. For tackling such problems, we plan to can couple our method with a strong visual perception module moving it to a direct perception approach.

VI. Acknowledgment

We thank Phaniteja S. from IIIT-Hyderabad and Anirban Santara from IIT-Kharagpur for their support & knowledge.

References

[1] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems (NIPS)*, 1989.

[2] Urs Muller, Jan Ben, Eric Cosatto, Beat Flepp, and Yann L Cun. Off-road obstacle avoidance through end-to-end learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2006.

[3] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[4] Jan Koutník, Giuseppe Cuccu, Jürgen Schmidhuber, and Faustino Gomez. Evolving large-scale neural networks for vision-based torcs. In *International Conference on the Foundations of Digital Games (FDG)*, 2013.

[5] Jiakai Zhang and Kyunghyun Cho. Query-efficient imitation learning for end-to-end autonomous driving. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2017.

[6] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[7] Mohammed Al-Qizwini, Iman Barjasteh, Hothaifa Al-Qassab, and Hayder Radha. Deep learning algorithm for autonomous driving using googlenet. In *IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2017.

[8] Bernhard Wymann, Eric Espié, Christophe Guionneau, Christos Dimitrakakis, Rémi Coulom, and Andrew Sumner. TORCS, The Open Racing Car Simulator. `www.torcs.org`, 2014.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*, 2013.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

[11] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016.

[12] Daniel CK Ngai and Nelson HC Yung. Automated vehicle overtaking based on a multiple-goal reinforcement learning framework. In *IEEE Intelligent Transportation Systems Conference (ITSC)*, 2007.

[13] Daniele Loiacono, Luigi Cardamone, and Pier Luca Lanzi. Simulated car racing championship: Competition software manual. *arXiv preprint arXiv:1304.1672*, 2013.

[14] Han-Hsien Huang and Tsaipei Wang. Learning overtaking and blocking skills in simulated car racing. In *IEEE Conference on Computational Intelligence and Games (CIG)*, 2015.

[15] Wei Xia, Huiyun Li, and Baopu Li. A control strategy of autonomous vehicles based on deep reinforcement learning. In *International Symposium on Computational Intelligence and Design (ISCID)*, 2016.

[16] 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.

[17] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017.

[18] Yan-Pan Lau. Using keras and deep deterministic policy gradient to play torcs. `yanpanlau.github.io/2016/10/11/Torcs-Keras.html`, 2016.

[19] Larry D Pyeatt and Adele E Howe. Learning to race: Experiments with a simulated race car. In *FLAIRS Conference*, 1998.

[20] Stuart Russell. Learning agents for uncertain environments. In *Conference on Computational Learning Theory*, 1998.

[21] Markus Wulfmeier, Dominic Zeng Wang, and Ingmar Posner. Watch this: Scalable cost-function learning for path planning in urban environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[22] Sahand Sharifzadeh, Ioannis Chiotellis, Rudolph Triebel, and Daniel Cremers. Learning to drive using inverse reinforcement learning and deep q-networks. In *NIPS workshop on Deep Learning for Action and Interaction*, 2016.

[23] 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. In *International Conference on Learning Representations (ICLR)*, 2016.

[24] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *International Conference on Machine Learning (ICML)*, 2009.

[25] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, 2015.

[26] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine Learning*, 1992.

[27] Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

[28] Naoto Yoshida. Gym-torcs. `github.com/ugo-nama-kun/gym_torcs`, 2016.

[29] George E Uhlenbeck and Leonard S Ornstein. On the theory of the brownian motion. *Physical review*, 1930.