Online Reinforcement Learning for Autonomous Driving

Gagan Jain, IIT Bombay [1] **Utkarsh Agarwal, IIT Bombay** [2] **Shubham Lohiya, IIT Bombay** [1]

Abstract

This paper attempts to solve the autonomous driving problem for an ideal, annotated driving and obstacle-free environment using an online Reinforcement Learning (RL) based approach. The self-vehicle is assumed to be equipped with sensors that provide visual information about the environment using a dashboard camera. The Q-learning algorithm is used to train the agent online using the CARLA interface, by discretizing the state space constructed using the visual input, with the help of tile-coding.

Starting with a brief introduction and the problem description with an accompanying literature review, this paper describes the approach used in detail along with the experiments performed and concludes with a high level discussion on potential improvements based on the results.

1. Introduction

Autonomous driving has always been an audacious visionary goal to human-kind. Today we are closer to achieving it than ever before, thanks to Artificial Intelligence and Robotics' continuously evolving fields and the improved computational capabilities. Researchers from all over the world are continually trying out new approaches to make this dream come true. While most techniques look at a problem from a classified point of view, there have been some attempts to find end-to-end solutions.

RL is considered one of the most substantial AI paradigms, for the fact that it can be used to teach machines through interaction with the environment and learn from their mistakes just like humans do. Despite its perceived utility, it has not yet been successfully applied in automotive applications. Motivated by the success of reinforcement learning on Atari games and Go, we propose a framework for autonomous driving using online reinforcement learning. This is of particular relevance as it is difficult to pose autonomous driving as a supervised learning problem due to strong interactions with the environment, including other vehicles, pedestrians, and roadworks.

This paper presents an online learning approach to the task of autonomous driving. While an online approach has the perk of being able to learn simultaneously while driving, but for the same reason, the computational requirements has to be kept under a certain limit to be able to synchronously learn and act. There has been a lot of research on the application of RL algorithms for the task of automated driving, however the approaches followed were majorly offline ones, with learning updates only at the end of the episode, and relatively lesser work has been done for the application of online methods for the same. Moreover, a significantly large fraction of the work done in this domain targets the problem of lane driving, wherein the sole objective of the agent is to minimize the time of travel while being allowed to go off-track at times.

This paper aims to be able to perform the task of self-driving in a more constraint city-like environment, with the requirement of always moving in the correct lane. The results indicate that even with a surprisingly low-dimensional feature space, the RL based agent is able to learn to drive to a good extent under the influence of an appropriately chosen reward function. In accordance with the arguments above, the upcoming sections present a comprehensive literature review followed by brief description of the problem statement.

2. Related Works

Various approaches have been taken taken to tackle the non-trivial task of Autonomous driving.

Imitation Learning is one of the earliest approaches to vision-based driving and has been quite successful. Recent works have extended imitation learning approaches to navigation in complicated urban environments. These methods train on trajectories (Codevilla et al., 2018; Dosovitskiy et al., 2017b; Pomerleau, 1989) or rich sensory data (Chen et al., 2020; Pan et al., 2017) collected by human experts. Hence they are limited to the expert's observations and actions.

Model-based reinforcement learning approaches model the environment to help train the policy. (Sutton, 1991), Kalweit and Boedecker (Kalweit and Boedecker, 2017), Gu et al. (Gu et al., 2016) use a forward model to generate imagined trajectories. Feinberg et al. (Feinberg et al., 2018),

[1]Department of Mechanical Engineering, IIT Bombay
[2]Department of Computer Science and Engineering, IIT Bombay.

Buckman et al. (Buckman et al., 2018) roll out forward models over short horizons to improve Q-function or value function approximation. Chen et al. (Chen et al., 2021) factor the forward world model into the controllable agent and a passively moving environment.

Deep Reinforcement Learning based approaches for autonomous driving and navigation have recently shown a lot of success with end-to-end training. Wolf et al. (Wolf et al., 2017) used a Deep Q Network to learn autonomous steering in simulation with discrete action space. Lillicrap et al. (Lillicrap et al., 2015) developed a continuous control algorithm capable of learning a deep neural policy to drive the car on a simulated race-track. Chen et al. (Chen et al., 2018) proposed a hierarchical deep RL approach capable of solving complex temporal delayed tasks.

We have developed a novel model-free, vision-based approach which uses tile-coding and Q-learning to tackle the lane following task. Our approach allows for online learning and is relatively computationally inexpensive.

3. Problem Formulation

Speaking on a very broad scale, we wish to solve the autonomous driving problem for an ideal, annotated driving environment. An ideal, annotated environment will have all the roads are appropriately marked with lanes, with all the traffic signs in place, and other vehicles as a part of the environment behave in a usual cooperative manner. The self-vehicle is assumed to be equipped with sensors that provide visual information about the environment.

Specifically here, we have used the CARLA Simulator (version 0.9.9.4)(Dosovitskiy et al., 2017a), to simulate the model of a small town with multiple double lane roads running across it. The CARLA Simulator allows us to spawn a client vehicle on the server which can be controlled using a throttle and a steering wheel and the motion states which change following the laws of physics can be observed. CARLA supports a range of different sensors like an RGB Camera, a depth Camera, Semantic segmentation camera, LiDAR, Radar, Collision Sensor, etc. that can be attached to the vehicle wherever we want to observe the surrounding in a realistic manner.

We here use the Semantic Segmentation Camera attached to the dashboard of the vehicle to observe the path in the front of the vehicle. the Semantic Segmentation Camera sensor functionality provides an $carla.Image$ object with the tag information encoded in the red channel. This is converted with the help of $CityScapesPalette$ in $carla.ColorConverter$ to apply the tags information and show picture with the semantic segmentation. Now, knowing the color of the road's encoding gives us the precise segmentation for it and we can perform feature extraction on it.

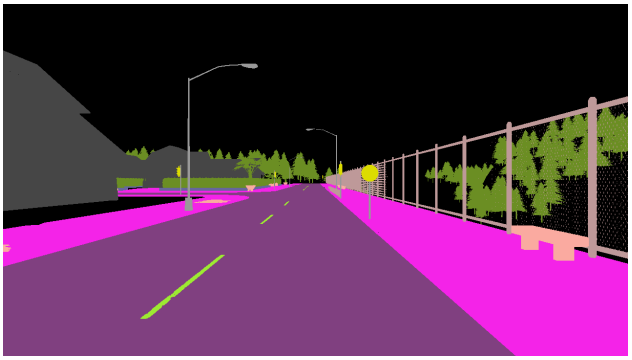

Figure 1. The Semantic Segmentation Dashboard Camera Feed

Other features such as vehicle speed, throttle, steer angle are directly subscribable from the simulator and are thus available. The state spaces are thus consistent with the data that a normal car on a road can access. We aim not to use any information that is not observable from a real life parallel.

We will be using just the dashboard camera feed for providing visual inputs for planning and control. The speed and steer angle in carla work almost similar to real life scenarios. On pressing the throttle, the velocity of the vehicle increases incrementally up to the time the throttle is pressed and begins to decline at a similar rate when the throttle is not pressed. Similarly, the steer angle changes incrementally proportional to the duration of time that a steering command is applied and slowly comes to 0 degrees if there is no steering control supplied.

We plan to train the vehicle agent incrementally by increasing the toughness of the task expected gradually. Hence, we begin with training the vehicle on a perfectly straight road. We want that the agent is successfully able to follow the road without colliding with anything or crossing any lines. The agent must stay in the correct lane and be able to move parallel to the road. We have used the 'Town01' for our simulation as it offers the least complex system of roads to manoeuvre around. We chose a spot with the longest straight path in front of it to train here. We spawn the vehicle here ($x = 2.0$, $y = 315.0$, $z = 2.0$, $yaw = 270.0$) every time the vehicle crashes or crosses a line and a new episode needs to be started. The non-zero value of z coordinate is provided to avoid collision of the agent with the road at spawn. The vehicle thus falls a little for a few instants whenever it is re-spawned. We also introduce some random pertubation to the x and yaw values so that the vehicle can not just quickly learn to just move straight but also has incentive to tune the steering ability to avoid crossing road lines or running over footpaths.

When the agent learns to ace this very well, we expose it

to the turns present in the map. We spawn the agent right before a turn which is not a T-point or a crossing but a simple right or a left turn. In this manner we would expose the agent to more and more complex scenarios and expect that it does learn to stay on the road and not collide or cross any lines. We would associate a large negative reward for such crashes and give positive reward primarily for covering a large distance on the road.

During this phase, we do not introduce any other vehicles that the agent may have to avoid or coordinate with nor any pedestrians. We are not taking regard of any speed limits or traffic signals either but expect the vehicle to move at a mid range optimal speed and not be very slow or extremely fast. We would also like that the effect of steering are also gradual and not very jerky.

4. Approach

The raw input from the dashboard camera is a semantically segmented image, which when directly used for learning will lead to a very high dimensional as well as redundant state-space representation, which will result in negligible amount of learning even after a lot of training. So, it becomes vital to reduce the complexity of the problem to a large extent.

For this, we use the semantic segmentation image provided by the dashboard camera to generate a set of features, which along with velocity, throttle and steer data is used to construct the state space. This state space is still continuous and thus, in order to discretize it, we use tile coding (Sherstov and Stone, 2005). This low-dimensional discretized state space is then used by the RL algorithm Q-learning (Watkins and Dayan, 1992), which makes online updates after every frame update in the CARLA simulator.

4.1. Feature extraction

The aim of the sensor inputs to the driving agent is to be able to properly represent important information about the environment. One way to do this is to directly use the raw sensor data such as the dashboard camera image. However, the detailed CARLA environment leads to the raw data containing of very high dimensional information, wherein the input image at every frame has a dimension $(720, 1280, 3)$, also including details like weather information and road texture. Working with such data would require immense amount of training so that every dimension of the state space is properly learned and the agent performance can be generalised to unseen scenarios.

In the past, learning based approaches have been accompanied with a perception module to determine the distances of the self-vehicle to certain fixed points in the camera frame. This information, when compactly represented, form very effective state space features. One way to do this is to detect the lanes and roads precisely using a bird-eye's view using Inverse Perspective Mapping (IPM) on the raw input image, which makes the distance computation mentioned above much easier, but as it turns out, this method requires an accurate knowledge of the location of the camera relative to the ground and is very sensitive to errors.

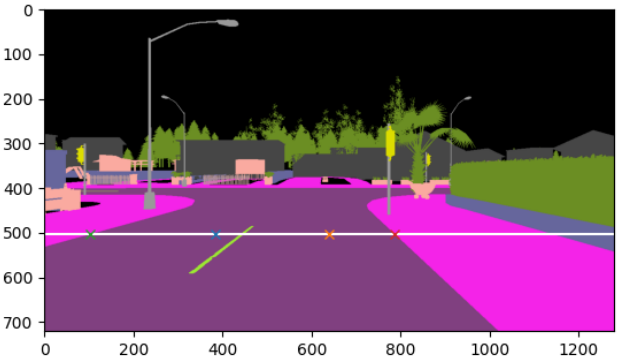

Figure 2. Determining left and right distances for feature extraction. The green and red markers represent the road boundaries and their distance respectively from the left and right edges of the frame is computed and normalized using the respective left and right distances of the blue and orange markers.

To address this problem, we instead use a simple image processing technique to determine two values. We construct a horizontal line [Fig. (2)] at a certain fixed location in the image for every frame, mark the point of intersection of the left (right) edge of the road and the horizontal line, and compute the distance of the left (right) point from the leftmost (rightmost) point on the horizontal line in the frame. This is further normalised to a value between 0 and 1 based on two limiting markers for the left distance and right distance.

Other than the two distances, we obtain the velocity, throttle and steer value during driving and treat them as parts of the state. The precise location of the car is also obtained (equivalent to having the car equipped with HD Maps/GPS) which will be used for reward modelled as explained in Section 5.

4.2. State encoding

After feature extraction, we restrict our state space input to 5 dimensional data consisting of left distance, right distance, velocity, throttle and steer. Note that while throttle and steer are actually control inputs, but their value at a particular instant is part of the state and it is the change in throttle and steer that govern the controls of the car.

While the representation complexity has been substantially reduced, still considering the fact that all five of these are

continuous variables, there is a need for discretization as the tabular method Q-learning would require a finite state space. For this purpose, we use 1-dimensional tile coding , in which each state is representation in terms of the discretized tiles they belong to across multiple tilings. The width and the offset of the tiles are hand-picked in order to make the discretization as uniform as possible. Tile coding has been found to be convergent with the algorithm Q-learning, thus the choice. Fig. 3 represents the discrete tilings created for discretization. This discretized representation is used as the state space for the problem.

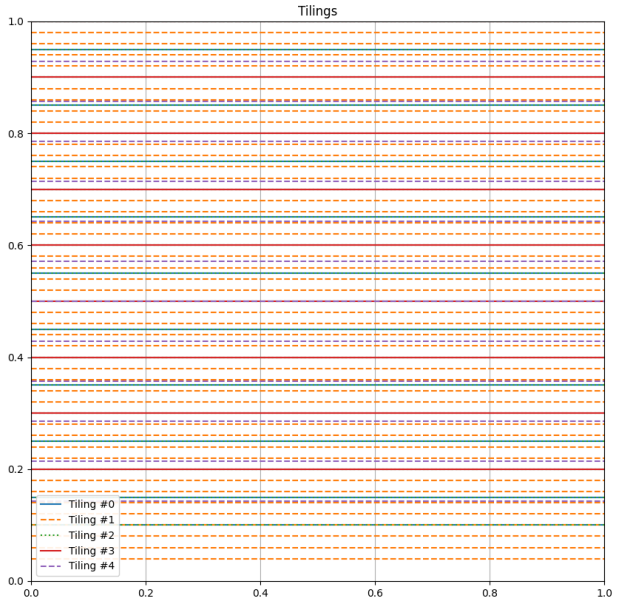

Figure 3. Visual representation of the five tilings

4.3. Q-learning

With access to the state *s* and reward function *r*, the objective of reinforcement learning is to find the optimal policy π^* that optimizes the expected future total reward. In order to find the optimal policy for the discretized state space, we use the Q-learning [1] algorithm.

The fact that Q-learning can be used as an online algorithm and is very computationally efficient allowed us to make the agent learn real-time at a good enough frame rate.

5. Experiments and Results

5.1. Reward Function

We have modelled the reward in 4 parts. Three of them are there to regulate the state values of velocity, steer angle and throttle, while the fourth component deals with giving

Algorithm 1 Q-learning

Parameters: step size $\alpha \in (0, 1]$, $\epsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$.
for each step of episode **do**
  Choose A ϵ-greedily from S using policy derived from Q
  Take action A, observe R, S' from CARLA Environment
  $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
  $S \leftarrow S'$
end for

a positive reward for distance covered. An additional large negative reward is given whenever there is a collision or a line cross, essentially when the vehicle wanders off-road. Each of the rewards are combined together by taking a weighted summation, the weights for which were decided over multiple iterations to be able to capture the optimal behaviour appropriately.

The first component corresponds to the velocity off the vehicle. We form a trapezoidal reward function to reward velocity values between 10 and $30km/h$. Another fixed negative reward is added to prevent very low values of velocity so that the agent not just stop fully.

The second component is to regularise values for small values of steer by a small amount. This is to ensure that the steering angle does not oscillate near the small values.

The third component of reward function deals with the throttle values. We penalise large values of the throttle as well as near zero values for it.

For the distance component, we provide a positive reward whenever the agent moves a certain fixed distance. Here, we provide a reward of 15 for every 10 metres travelled. Here, loc1 is initialised at every agent restart. In the game loop, loc2 keeps updating and stored the latest position of the vehicle. Whenever the distance between, loc1 and loc2 exceeds 10 metres, we give the reward and update loc1 as loc2.

5.2. Training Details

The training is carried out with the following set of parameters for Q-learning:

$$\alpha = 0.1, \epsilon = 0.9 \tag{1}$$

The five set of tilings have been provided the widths and offsets as shown in Table 1.

The training was done for a total of 257 episodes with a total training time of approximately 4 hours on the GPU

Algorithm 2 Reward Function

```
rewardV = 0, rewardT = 0, rewardS = 0

velFac = 0.02
if vel < 3 then
    rewardV += -0.02
else if vel < 10 then
    rewardV += velFac*vel
else if vel < 30 then
    rewardV += velFac*10
else
    rewardV += velFac*(40-vel)
end if

steerFac = 5.0
if abs(steer) < 0.3 then
    rewardS -= steerFac*(steer**2)
end if

throttleFac = 3.0
rewardT -= throttleFac*((throttle-0.2)^2)
if throttle > 0.5 then
    rewardT *= (throttle+0.5)
end if
if throttle < 0.2 then
    rewardT *= 2
end if

if distance(loc2, loc1) > 10 then
    reward += 15
    loc1 = loc2
end if
```

NVIDIA GeForce GTX 1650 Max-Q and the results are as shown in the next subsection.

5.3. Results

After training on the CARLA environment, the agent was able to successfully drive without collisions and along the track of 300 metres staying in the correct lane at all times. The total reward accumulated is shown below, along with the individual reward contributions from the four components of the reward function. Note that the most significant positive reinforcement comes from the distance reward, which incentives the agent to keep covering more distance. At the same time, the negative throttle reward is kept such that the throttle is kept as a positive but low value for most times, which makes it easier to control the vehicle in case of sudden changes in the state.

Table 1. Tile coding details

TILING	WIDTH	OFFSET
1	10	0.1
2	10	0.23
3	30	0.03
4	10	0.42
5	14	0.14

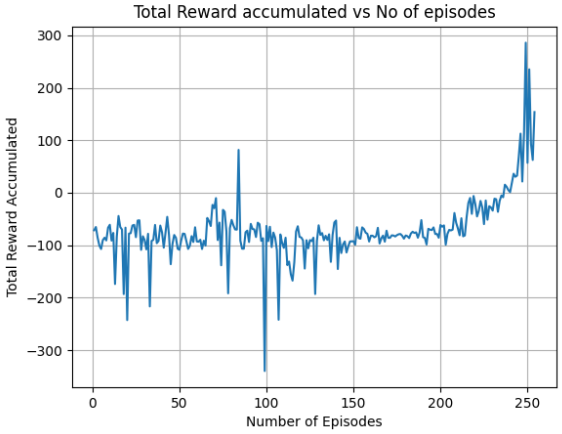

Figure 4. Total Reward

6. Discussion and Future Work

The project has been a successful demonstration of the fact that online learning algorithms like Q-learning can be used to make the agent learn to drive in an ideal city environment. The Town 01 environment in CARLA is a fairly complicated one, and while the initial plan was to construct a map on our own which would have been relatively easier to work with and then move on to this environment, we figured out that the RoadRunner package in CARLA required us to build it from source, requiring 70 GBs of download. So, we directly started working with this map and it turned out fairly well. There are several improvements and developments that can be done taking this project forward, some of which include:

- While the agent was able to deal with small turns and straight roads impeccably after training, it still was unable to drive on regions with very sharp turns. Through inspection, it was clear that the agent was learning and given more training time, the agent might be able to perform better. The reward function could have been modelled better with special consideration for turns beyond a certain angle.
- The state space representation was a fairly straightforward one. More features can be introduced in the

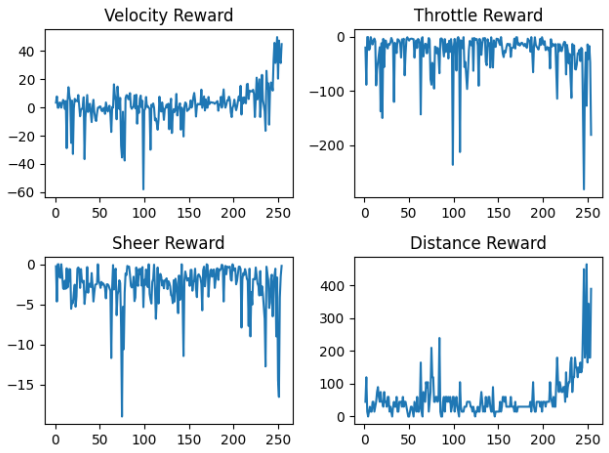

Figure 5. Contribution of Individual Reward Components

representation, which might be able to improve the agent's performance even further.

- We played with creating three road masks for detecting the possible turns ahead. One for detecting if there is a way straight and similarly two for right and left turns. We just had to take a dot product of the road and the mask and perform thresholding on the returned values to determine the turns. This will help the agent get to understand and choose a path based on the choices it has for a multi-turn. Since we could not get to tackling a T-point, we have not used this function.

- In this project, we have demonstrated the capability of autonomous driving using Q-learning in an obstacle-free environment. In future, a more complicated perception module can be used to be able to deal with dynamic obstacles in the environment like other vehicles, pedestrians, etc.

- We have neglected the existence of traffic regulations like speed limits and traffic signals in this project, but this can be easily incorporated and worked with by appropriately detecting and decided on the controls should such a situation arise.

- While Q-learning has shown some promising results, it will be interesting to try Deep Reinforcement Learning methods for the same and the future tasks. The major barrier in this will be of computational complexity as the model will have to be trained online with increasing data. A right balance between model complexity and performance expectations need to be met here

References

Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. *arXiv preprint arXiv:1807.01675*, 2018.

Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Conference on Robot Learning*, pages 66–75. PMLR, 2020.

Dian Chen, Vladlen Koltun, and Philipp Krähenbühl. Learning to drive from a world on rails. *arXiv e-prints*, pages arXiv–2105, 2021.

Jianyu Chen, Zining Wang, and Masayoshi Tomizuka. Deep hierarchical reinforcement learning for autonomous driving with distinct behaviors. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1239–1244. IEEE, 2018.

Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4693–4700. IEEE, 2018.

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, 2017a.

Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017b.

Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I Jordan, Joseph E Gonzalez, and Sergey Levine. Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101*, 2018.

Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838. PMLR, 2016.

Gabriel Kalweit and Joschka Boedecker. Uncertainty-driven imagination for continuous deep reinforcement learning. In *Conference on Robot Learning*, pages 195–206. PMLR, 2017.

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.

Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos Theodorou, and Byron

Boots. Agile autonomous driving using end-to-end deep imitation learning. *arXiv preprint arXiv:1709.07174*, 2017.

Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA ARTIFICIAL INTELLIGENCE AND PSYCHOLOGY . . . , 1989.

Alexander A Sherstov and Peter Stone. Function approximation via tile coding: Automating parameter choice. In *International Symposium on Abstraction, Reformulation, and Approximation*, pages 194–205. Springer, 2005.

Richard S Sutton. Planning by incremental dynamic programming. In *Machine Learning Proceedings 1991*, pages 353–357. Elsevier, 1991.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Peter Wolf, Christian Hubschneider, Michael Weber, André Bauer, Jonathan Härtl, Fabian Dürr, and J Marius Zöllner. Learning how to drive in a real world simulation with deep q-networks. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 244–250. IEEE, 2017.