# Deep Q-Learning Applied to Obstacle-avoiding Actions of Self-driving Vehicles

Yixian Li[1], Junnan Yu[2], and Xiaohan Song[3]

[1]s031820@stanford.edu
[2]jy52168@stanford.edu
[3]xhsong@stanford.edu

## I. Abstract

In this project, an online deep-Q learning algorithm is implemented on a vehicle model that finally becomes able to avoid obstacles and cross a goal line from a given starting position. The team makes different designs of exploration policy, neural network structure, and reward function. With various settings of these key components, numerous experiments are conducted in a virtual environment. As a result, changing the design of each component has its unique impact on whether the vehicle can be successfully trained or not and the stability of its learning process.

## II. Introduction

### A. Motivation and Background

Autonomous vehicles, because of their reduced driver effort and low carbon emission properties, have been put in the center of spotlight. Nevertheless, they also bring great concern since failing to recognize and avoid obstacles causes safety threat to human drivers and lead to infrastructure damage. In this project, we aim to optimize the performance of a self-driving vehicle in an obstacle-existed environment. Specifically, we desire to reach the goal line while successfully avoiding all the obstacles in a highway scenario. No external sensor is installed on the vehicle, so it does not receive any feedback from the environment but only updates parameters by exploration. Experiments are conducted in a virtual environment, in which the locations of obstacles are predetermined but not given to the vehicle. The environment setup is presented in section IV-A.

### B. Problem Details

*1) State Space:* Assuming a constant linear velocity, we defined the state space $\{s\}$ of the vehicle as a 4-dimensional vector denoted as:

$$(x,\ y,\ \theta,\ \omega)$$

where $x$ is the x position, $y$ is the y position, $\theta$ is the heading angle, and $\omega$ is the angular velocity.

*2) Action Space:* Our action space has 3 actions represented by 2-dimensional tuples that consist of an angular acceleration and a linear acceleration. Angular acceleration is set to be 0.2, -0.2, or 0. The linear acceleration is always 0 because we assumed constant linear velocity. Each tuple is accordingly denoted as

$$(\alpha, a)$$

where $\alpha$ is the angular acceleration, and $a$ is the linear acceleration. Hence, we have three actions: (0.2, 0) represents left steering, (-0.2, 0) represents right steering, and (0, 0) represents no steering.

*3) Evaluation Metrics:* In order to evaluate model performance, we devise a qualitative metric recorded as whether the car successfully passes the goal line or not at the end of training episodes. We also set a quantitative metric for the models that do finally achieve this objective, which is the accumulative reward collected by the vehicle using the same reward function throughout the training episodes. Details of reward function are presented in section V-D.

## III. Related Work

### A. CARLO

We are using *Carlo* [1], a 2-dimensional driving simulator developed by Stanford ASL, to tackle this problem. The environment includes various self-defined agents, dynamics and interactions. Researchers could develop upon the environment with various reinforcement learning and imitation learning tasks.

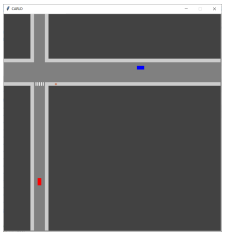

Fig. 1. *Carlo*, the 2-D Driving Simulator

### B. Deep Q-Learning

Q-Learning [2] is a model-free reinforcement learning method initially proposed by C.J.C.H Watkins. The method incorporates the idea of incremental estimation of mean with the representation of Q function in reinforcement learning context. The algorithm we used, deep Q-learning proposed by *DeepMind system* [3], is a variant of Q-learning where we estimate our Q function via neural network to account for a high-dimensional state space.

In the project, we adapted our code from a particular source in *LiveLesson* [4].

### C. Double deep Q-Learning

Besides the deep Q-Learning method we adopted, we also reviewed the deep reinforcement method via double Q-learning [5]. Compared with deep Q-learning, deep double Q-learning addresses the overestimation issue from deep Q-learning by proposing a learning mechanism that uses two neural networks of same structure to estimate two separate value functions. The difference is that one's discounted future is evaluated using the other one's policy. The article claimed that this method outperforms deep Q-learning in a game scenario. Because of extra computation effort from neural network, we would attempt this method in future work.

### D. Deep Actor-Critic Method

Actor-critic methods [6] are on-policy reinforcement learning methods. They are temporal difference methods that have a separate structure to explicitly represent the policy independent of the value function. To account for both in large state space setting, two neural networks will be applied. The advantage of using the actor-critic method is it takes the benefit from using both value-based and policy-based learning, yet still introduce extra hyper-parameters for design and potential computational cost.

## IV. Environment Setup and Data Acquisition

### A. Environment Setup

The environment setup in Carlo is visually described in Fig.2

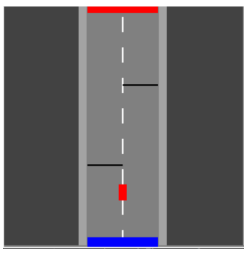

Fig. 2. A 60 by 60 Carlo environment. Being designated to start from the center point of the blue region, the vehicle gets reset once it hits an obstacle (marked as black lines) or the sidewalls. It keeps updating the choice of action along its way in order to approach the red goal line. The car moves with a constant linear velocity of 5.

### B. Data Acquisition

We collected our data by running online simulations based on a proper exploration policy described in section V-B. The data collected at each time step is in tuple form $(s,a,r,s')$, where $s$ denotes the current state, $a$ denotes the current action, $r$ denotes the current reward, and $s'$ denotes the resulting state. We collected this data with a time interval $dt = 0.3$ seconds.

## V. Methods

### A. Deep Q-learning Algorithm

In the context of Q-learning, we want to evaluate the Q-value of a certain state and action pair, which are later used to determine the best control policy given a certain state. The Q-value is defined as:

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} T(s'|s,a) \max_{a'} Q(s',a')$$

This could also be represented by the expected value:

$$Q(s,a) = \mathbb{E}_{r,s'}[r + \gamma \max_{a'} Q(s',a')]$$

where $r$ is the immediate reward. With knowledge of $s$, $a$, $r$ and $s'$ of each example, we could perform an update on $Q(s,a)$:

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

where $\alpha$ is the learning rate.

The Q value will eventually converge with sufficient number of iterations on the data set, where the difference between $r + \gamma \max_{a'} Q(s',a')$ and $Q(s,a)$ goes to zero.

Because our problem has a large state space, we need to fit a model to approximate $Q(s,a)$. This is done by using a neural network described in section V-C. A batch of examples are fed into the neural network, and loss function would be constructed by minimizing the mean square error (MSE) that represents the difference described above. After successfully training the network, we could retrieve the optimal action of a state $s$ by:

$$\pi = \arg\max_{a} Q(s,a)$$

### B. Exploration Policy

We collect our data in tuples $(s,a,r,s')$. In an episode, we continue to collect data until the vehicle reaches the finishing line, hits the obstacle, or exceeds the time limit. The action for every time step in the episode is chosen based on our exploration policy described in the later section. If an episode ends, the simulation is reset and run again. In addition, we will use a memory that holds 2000 data at maximum. We drop the older data to incorporate new data that has more valuable information. A batch of $n$ data tuples are sampled from the memory and then fed into the neural network for training.

We utilized the decaying epsilon-greedy method for the exploration strategy. This is described as:

$$a = \begin{cases} \text{random action} & \text{for probability } \varepsilon \\ \text{argmax}_a Q(s,a) & \text{for probability } 1-\varepsilon \end{cases}$$

We start with $\varepsilon = 1$ and steadily decay $\varepsilon$ as more data is collected. A decay factor $d$ and a lower bound $\underline{\varepsilon}$ are introduced. Hence, after each iteration, $\varepsilon$ is redefined as:

$$\varepsilon := \max\{d\varepsilon, \underline{\varepsilon}\}$$

### C. Neural Network Design

As shown in Fig.3, the output of the neural network is the predicted Q-value for all 3 actions (output size is 3) corresponding to the input state (input size is 4).

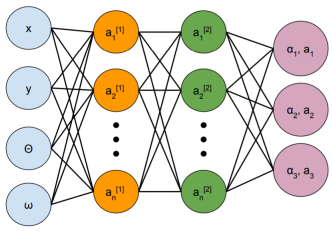


Fig. 3. A neural network of deep Q-learning with 4 inputs and 3 outputs. This particular design incorporates two hidden layers.

In the project, a 3-layer neural network with ReLU activation in its hidden layers is algebraically defined as:

$$a^{[1]} = \text{ReLU}\left(W^{[1]\top} s + b^{[1]}\right)$$
$$a^{[2]} = \text{ReLU}\left(W^{[2]\top} a^{[1]} + b^{[2]}\right)$$
$$\hat{y} = W^{[2]\top} a^{[2]} + b^{[3]}$$

where output $\hat{y}$ is a 3-dimensional vector, and its $i$th entity represents the corresponding Q-value that has been learned, which is $Q(s,\hat{y}_i)$.

We use mean squared error (MSE) as the cost function. From section A, we know that if we try to optimize $Q$(s,a) toward convergence, the error we need to minimize is:

$$(r + \gamma \max_{a'} Q(s',a')) - Q(s,a)$$

In other words, we want to optimize $\hat{y}_i = Q(s,a)$ based on the truth value $y_i = r + \gamma \max_{a'} Q(s',a')$. MSE is calculated by:

$$\frac{1}{n}\sum_{j=1}^{n} (\hat{y}_i^{[j]} - y_i^{[j]})^2$$

where n is the batch size, j is the index of the training example.

### D. Reward Function

The reward r from each state is defined as follows:

$$r = \begin{cases} -200 & \text{collision exists} \\ +200 & \text{goal reached} \\ r_s & \text{otherwise} \end{cases}$$

where $r_s$ is the reward shaping term that encourages the car to move straight forward.

The design of our $r$ puts some abrupt value changes on the boundaries of the obstacles and destinations, but $r_s$ is also important, which gives us different step reward according to the state we are at. An example of a reward function we used is shown in Fig. 4.

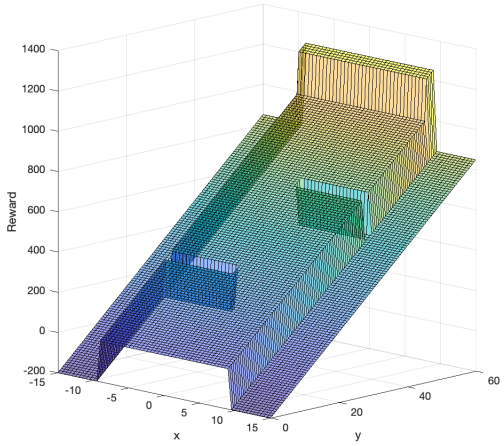


Fig. 4. A choice of reward function with a reward shaping of $r_{s2}$, which is later introduced in section VI-A

## VI. Experiment and Result

### A. Hyper-parameter

To simplify the problem, limit the ability of our vehicles changing its state, and expedite the calculation, we set our time step to be 0.3.

To make sure that the vehicle will explore enough successful routes before limiting its random exploration ability, we set the decay parameter $d$ to be 0.998 and decaying $\varepsilon$ to $\underline{\varepsilon} = 0.01$. The maximum episode for our training simulations is 3001.

To show the importance of the added reward shaping in obtaining successful training, we have also trained our vehicles under different reward shaping methods. We first tried out a reward function without reward shaping (i.e. $r_s = 0$). We then explored two other shaping cases with:

$$r_{s1} = -\frac{(x - x_{center})^2}{16} + \frac{60}{(y - y_{goal})}$$

and

$$r_{s2} = 25\sin\theta - 1$$

where $x_{center}$ denotes the map center's x value, and $y_{goal}$ denotes the goal position's y value.

$r_{s1}$ is based on the rationale to punish the car from being away from the center line (otherwise it may hit the sidewall, which often happens) and reward the car for being closer to the goal. $r_{s2}$ is based on the rationale to reward the car for heading towards the destination every time step. Our experiment is conducted under the same 64×32 neural network structure.

To test the influence of the complexity of neural network structure on the training performance, we trained the vehicle under (i) 128 neuron single-layer network, (ii) 32×24 neuron double-layer network, and (iii) 64×32 neuron double-layer network. Within the neural network structure, we used ReLU as the activation function in each of the hidden layers for saving computational effort. They are linearly connected to the 3 outputs as we are learning a Q value. The learning rate is set to 0.001. A batch size of 32 is used. The discount factor $\gamma$ is set to be 0.95. These knowledge mainly comes from various experiments conducted in reinforcement learning setting. For this portion of experiment, we fixed the reward function with a reward shaping of $r_s = 25\sin\theta - 1$.

### B. Results and Discussions

We first explored the effect of reward function on training quality under the same neural network structure. We used the metric introduced earlier in section II-B. With no reward shaping, the reward history and corresponding simulation result is presented in Fig 5 and Fig 6.

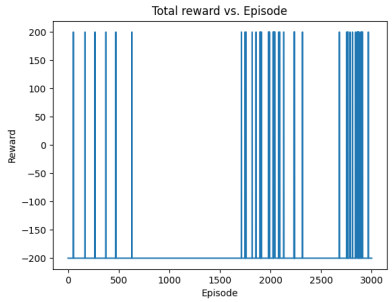


Fig. 5. The reward history under a 64×32 network without reward shaping

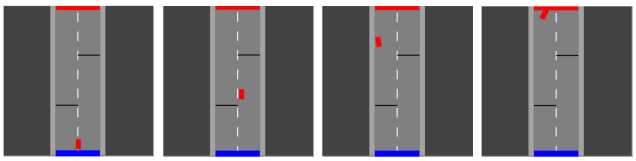

Fig. 6. The best simulation result under a 64×32 network without reward shaping. The weight is obtained from the last 200 training episodes

Because of the instability of deep Q-learning network that may cause variation in the learned policy, we chose the weight that gave us the best policy (successfully avoiding all obstacles and kept itself away from sidewalls) from the last 200 training episodes. Next, we explored the case with $r_{s1}$ as the reward shaping function. The results are shown in Fig 7 and Fig 8.

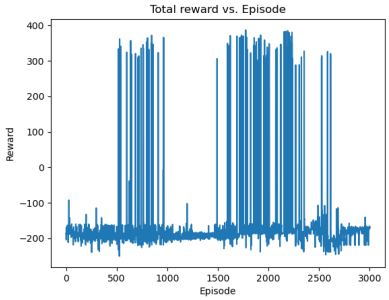


Fig. 7. The reward history under a 64×32 network with the reward shaping of $r_{s1}$

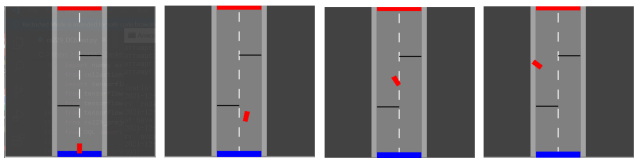

Fig. 8. The best simulation result under a 64×32 network with a reward shaping function of $r_{s2}$. The weight is obtained from the last 200 training episodes

Finally, we explored the case with $r_{s2}$ as the reward shaping function. The result is presented in Fig 9 and Fig 10.

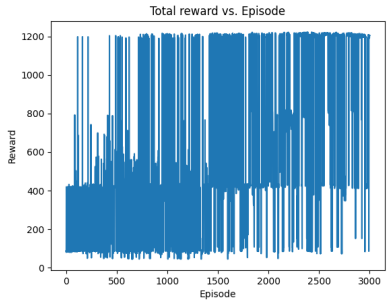


Fig. 9. Reward History under a 64×32 network with the reward shaping function of $r_{s1}$

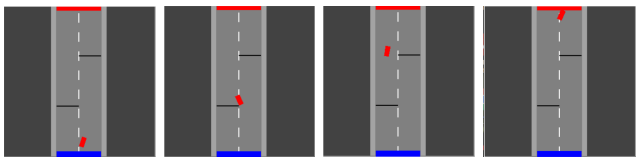

Fig. 10. The best simulation result under a 64×32 network with a reward shaping function of $r_{s2}$. The weight is obtained from the last 200 training episodes

We could not quantitatively compare the magnitude of total reward from the reward history due to different reward

function used. But the plots could be interpreted qualitatively by examining the convergence. With no reward shaping, the total reward takes only two values, -200 and 200. Indicated by the density of the lines, we see a convergence near the end of training, yet it is very unstable. With $r_{s1}$, the plot did not show convergence at the end. With $r_{s2}$, we start to see a clear convergence at around 1600 episodes, even though the plot is still oscillating due to the instability of the deep Q network.

The simulation result visually presents us with the quality of the policy. The best policy with no reward shaping case did successfully maneuver through the obstacles, yet failed to did that convincingly as it nearly hit the sidewall at the end. The best policy with $r_{s1}$ did not pass through the goal, whereas $r_{s2}$ passed through the goal with a convincing route.

After experimenting with various reward functions, we discovered that complicated reward reshaping may give us even worse performance than that of no reward reshaping, since it may potentially make it hard for the Q network to learn the appropriate Q value. In addition, for our problem, we found out that having the reward from an appropriate reward reshaping function is more dominating than the specified sparse reward/punishment and leads to better result. Seen in Fig 9, the highest total reward is around 1200 because the reward we collected at each time step overpowers the sparse reward from reaching the goal or punishment from hitting the obstacle.

We carried on our experiments with different neural network, using $r_{s2}$, qualitatively the best reward function. Under a 128-neuron single-layer network, the reward history is shown in Fig 11.

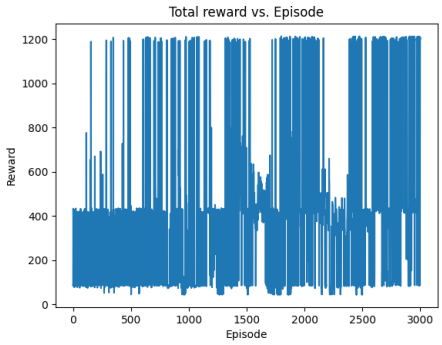


Fig. 11. The reward history under a 128-neuron single-layer network with a reward shaping of $r_{s2}$

Under a 32×24 neurons double-layer network, the reward history is shown in Fig 12.

Quantitatively, the maximum reward achieved from all three cases during 3000 episodes is nearly the same, indicating the vehicle did learn a decent policy based on the qualitative result for the 64×32 network case. However, qualitatively, the plot for 128 network did not convincingly converge at the end, as indicated by the trend. A 32×24 network is converging but still not stable, as with the case for 64×32 network, which oscillates frequently. The simulation visualization for using different neural network is not presented here, but they are all similar to the result shown in Fig 10.

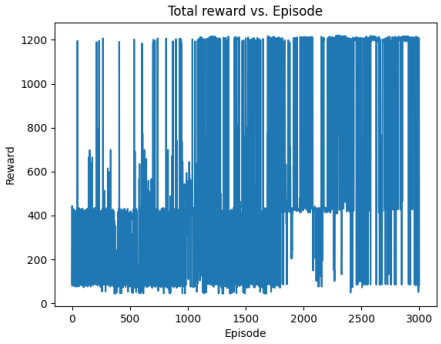


Fig. 12. The reward history under a 32×24 neurons double-layer network with a reward shaping function of $r_{s2}$

After using different neural network sizes, we observed that simpler neural network would not work well when tackling high dimensional state space problem with a combination of sparse reward and reward shaping. However, a complicated neural network would still suffer from instability. Once we explored a new state in the grid, it may be likely to bring up a considerable amount of weight change in the network that it needs couple of more episodes to adjust the policy back to the optimal. With this thinking, a more complex neural network may have higher instability due to potential over-fitting.

## VII. Conclusion & Future Work

In the project, we tackled the obstacle-avoiding problem in a 2-dimensional driving scenario with a certain complexity via deep Q-learning. Under a defined decaying-epsilon exploration policy, we explored the influence of different reward functions and network structures on the success of completing the task and resulting network stability. From the experiment, we concluded that implementing a reward shaping function that serves as dominating reward and the function not being too complicated is crucial to the quality of training result. In addition, the neural network size should be sufficiently large to enable convergence in learning, but we should not overly construct the network to create more instability.

The learning algorithm could be improved in terms of the efficiency and stability of the vehicle's learning process. We may observe better outcome by changing other hyperparameters including batch size, learning rate, and episode length. Furthermore, we could have a more reliable exploration policy and higher chance to pass through more complex distribution of obstacles if the vehicle is implemented with a sensor which gives live feedback from the environment. In addition, different algorithms presented in III would be attempted if given more time.

## VIII. Contribution

Yixian Li: Experiment environment setup; Deep Q-learning algorithm setup; Exploration strategy setup; Initialization of hyper-parameters.

Junnan Yu: Reward function refinement; Conducting experiments; Episodes training; Hyper-parameters tuning of cost function. Reporting and debugging errors.

Xiaohan Song: Neural network design; Reward function design; Hyper-parameters tuning of neural network; Loss function design.

Including trained neural network weight and codes, the github link for this project is: https://github.com/Yixian-work/cs229-project

## References


[1] E. Bıyık, *Stanford-iliad/carlo, a 2d driving simulator*, `https://github.com/Stanford-ILIAD/CARLO`, 2019.

[2] C. J. C. H. Watkins, "Learning from delayed rewards," 1989.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[4] livelessons, *Deep q learning networks*, `https://www.youtube.com/watch?v=OYhFoMySoVs`, Sep. 2018.

[5] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.

[6] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," *Advances in neural information processing systems*, vol. 12, 1999.