A study of multiple reward function performances for vehicle collision avoidance systems applying the DQN algorithm in reinforcement learning

N J Zakaria¹, M I Shapiai¹ and N Wahid²*

¹Center for Artificial Intelligence (AI) and Robotics Ikohza, Malaysia-Japan International Institute of Technology, Universiti Teknologi Malaysia, Jalan Sultan Yahya Petra, 54100, Kuala Lumpur
²Faculty of Electrical Engineering, Universiti Teknologi MARA, Cawangan Terengganu, 23000 Dungun, Terengganu, Malaysia

*Corresponding author: nurbaiti@uitm.edu.my

Abstract. Reinforcement Learning (RL) is an area of Machine Learning (ML) that intends to improve the acts of agents learning from environmental interconnection. The significant concern in RL is to achieve the promising potential of the training process in the model. However, network convergence speed is often sluggish in RL and converges quickly to local optimal solutions. Reward function has been used to deal with these problems as a useful tool to speed up the agent's learning speed. Even though RL convergence properties have been comprehensively explored, there are no specific rules for choosing the reward function. Therefore, searching for efficient potential reward function is still an exciting field of study. This paper discusses the reward function, execute some analysis, and provides the learning agent with the extracted information to increase the speed of learning for collision avoidance task. We provide an experimental study for selecting one reward function in a simulated collision-avoidance environment of an autonomous vehicle by applying the DQN algorithm. It has been conducted on online environments, which is using the CARLA simulator. This experimental study consists of three cases with a various exploration of reward values. Case 1 consists of the range of the penalty value larger than the reward function by 200 times. Case 2 is similar, but with the small range number of the penalty is applied, case 3, which is the reward function and penalty value, is in the same range value. The result shows that case 3 performances outperform case 1 and case 2 with 94% average accuracy; meanwhile, case 1 obtains 70%, and case 2 achieves 85% accuracy. It is may due to the monumental size of the collision penalty in comparison to all else. Hence, the findings obtained show the efficacy of the exploration of the reward function.

1. Introduction

Reinforcement Learning (RL) is one of the Machine Learning (ML) algorithms that acquires information based on environmental observation. Rewards are the consequence of learning in the environment. RL’s significant advantage is that, even though the environment is too broad or cannot be defined shortly, it generates the most successful rewards [1]. Nowadays, RL has currently achieved incredible performance in many applications [2]. RL aims to optimize an agent's action based on the environment's reward [1]. The current desirability of environmental states is denoted by reward. The state's value is the final reward
that an agent can determine to accumulate for future choices. The decisions are based on the agents' positive and negative outcomes. The agent performs an environmental reward action and adjusts its policy based on the reward. Through this continual interaction with the environment, the agent can learn the optimal policy to achieve the highest cumulative reward. The reward function thus implicitly determines the agent's optimal actions [3]. Q-learning [1] is a common type of RL where the optimal policy is implicitly learned in the form of a Q-function. Therefore, it is an RL method that works by learning an action-value function. The expected utility is obtained by a given action in a given state and following a fixed policy in the future. Q-learning can compare the predicted utility of behavior without having an environmental model. A recent variation in the Q-learning algorithm has shown a valuable enhancement in acquiring positive rewards [1].

However, slowness in convergence [4] is one of the key drawbacks of RL. Therefore, it is also challenging to develop a perfect reward function for a particular situation due to the environment's complexities [2]. Reward shaping is one of an approach to deal with this issue [3]. In this approach, the designer practically creates a virtual reward function and gives it to the learning agent. In the first episodes of learning from this virtual reward function, the agent receives the reward since it does not take any reward from the environment in the early episodes of learning. To optimize its behavior, the agent utilizes this virtual reward. Therefore, the shaped reward could have a vital effect on accelerating the training process. The main idea of reward shaping is to supply additional feedback by the designer, other than that of the environment, for the agent to enhance its convergence rate and training speed [3]. The agent changes its behavior in the following episodes based on the information gained from the environment, and the influence of the formed reward is gradually reduced. Although reward shaping has proven to be a powerful process, it is challenging and sometimes even impossible to determine the values of the shape rewards for extremely large environments.

Therefore, several methods to speed up RL have been proposed. M. J. Mataric [5] suggested a technique for developing reward functions that take advantage of implicit knowledge of the domain. It contains the use of continuous reward functions. E. Wiewiora [5] completed the reward shaping research and proved certain similarities between potential-based shaping and initial Q-values. Besides, with reward shaping, the rewards from the environment are augmented with additional rewards [6]. However, the reward shaping can lead the agent into learning suboptimal policies and thus trap the system. Indeed, selecting the initial Q-values [7] is the most basic approach for biasing training.

Besides, in RL, to adapt to the environment, an agent must interact with the environment, thereby forcing the agent to spend a large amount of time obtaining optimal behavior. In this process, the agent learns how to interact in the environment based on the rewards. To increase this learning process, it is hoped that a reward function can describe the agent’s state in a timely and accurate manner. Therefore, the design of a reward function has become an essential aspect of RL. To accelerate learning and training, researchers have proposed numerous methods to design reward functions. Wei et al. presented a study of heuristic reward function for RL algorithms [8]. Meanwhile, Yan et al. applied a hierarchical RL (HRL) algorithm based on a heuristic reward function to solve a vast state space issue [9]. Next, Maja et al. presented a technique for developing reinforcement functions that accelerate learning using implicit domain knowledge [10]. S. Koenig and R. G. Simmons studied different representations of reward functions and the sophistication of Q-learning techniques depending on the preference of RL representation. S. Behnke and M. Bennewitz proposed to give the learning agent access to the Q-values of the experienced agent concerning imitative reinforcement [11]. Mestol et al. presented an algorithm that is a variation of RL with the adaptation of reward to the degree of uncertainty of a performed prediction and solved some issues that could not be solved using conventional RL [12]. Next, Wu et al. suggested an adaptive network scaling framework to gain an appropriate scale of rewards during training for a promising performance [13]. In a system based on the concept of indirect reciprocity, Xiao et al. developed a technique for updating the node reward, which highly improved the system’s security [14].

Nevertheless, though RL has been extensively studied and its convergence properties are well known, individuals frequently prefer reward function on one's intuition [4] in practice. This paper presents an analysis of different variations in the reward function of the Q-learning algorithm to obtain better and
higher rewards. To propose a generic analysis, we address the effects of RL parameters on the policy. Therefore, we use the Q-learning algorithm to validate our analysis. The key challenge is to shed light on initializing RL parameters correctly to achieve the desired optimal behavior within the framework of a goal-directed task in a minimal time. Therefore, to obtain the optimum Q-value, the critical goal is to increase the efficiency of the Q-learning algorithm and to decrease the number of iterations. We herein concentrate on optimizing the reward function based on the existing reward function to produce better outcomes. By observing the changes in rewards during the RL process, especially when an agent succeeds or fails, we discovered that rewards often change significantly. Throughout this paper, we will focus on the case of three reward functions. We conclude by discussing how the analysis can be generalized to more than two reward functions. In this paper, the reward function consists of the values gained considering the speed of the agent in the environment.

2. Methodology

2.1. Reinforcement learning background

RL has seven crucial keywords: agent, environment, actions, states, episodes, rewards, and policy. RL process/tasks consist of training an agent which associates with its environment. By performing actions, the agent arrives at various scenarios known as states. Next, every action will give on to either positive or negative rewards. The only objective of the agent is to maximize its total reward over an episode. This episode is all events that occur within the environment between the first state and the last or terminal state. The designer reinforces the agent to learn from experience by executing the best actions called policy/strategy. Thus, the increases experience it gains, the highest it's to perform the action.

Figure 1. Agent-environment interaction loop.

Figure 1 shows the agent-environment interaction loop that consists of three components which are State ($s_t$), Action ($a_t$), and Reward ($r_t$). State ($s$) is some instance of the environment at the time ($t$). While Action ($a$) is what the agent does in the environment and Reward ($r$) is concerned about if the action is correct, the reward is high and vice versa. Therefore, the RL also contains two functions: Policy ($\pi$) and Value ($V$). Policy function is a function for choosing actions $a$ while Value function is a function for expected reward ($r$) given some action ($a$). The Policy function also decides what actions to take given the input state, while the Value function is the expected return: it notifies the designer how good that action within that state is.

Next, in the RL, the agent is made up of two components: memory and brain, as described in figure 2. Brain encompasses the Neural Network (NN). It is the house of AI systems. Here is the decision-making on what action to takes happens. Meanwhile, the memory stores the data from the previous state (reward + action). The data are used to train in the NN. For every thousand experiences/episodes, the NN gets better at performing specific actions. Eventually, it always performs the actions that always get really good rewards since they will always lead the agent to perform better for the next episodes.
2.2. Reinforcement learning problem formulation

Learning a policy function \( \pi \) to select the action

\[
a_t = \pi(s_t)
\]

Select actions that receive maximum reward \( \pi^* \)

\[
\pi^* = \arg\max \mathbb{E}[\sum_{t=0}^\infty \gamma^t r_t | \pi]
\]

Where \( \gamma^t \) is the discount factor, a value between 0 to 1. \( \mathbb{E}[\sum_{t=0}^\infty \gamma^t r_t | \pi] \) is known as Expected Return, i.e., \( V \). Thus,

\[
\pi^* = \arg\max (V^\pi)
\]

The policy function denoted by \( \pi \) is a decision-making function that decides what actions to apply. The optimal action is the action with the highest reward. To achieve the optimal action, it is necessary to obtain the optimal policy. So, the equation for the optimal policy, which is denoted by \( \pi^* \) is the maximum of the expected reward given that particular policy at that time. The \( \gamma^t \) is the discount factor. The discount factor is applying because it can't be sure either the reward that will be achieved in the future is as good as a reward earned in the current state. The value of the discount factor is set to 1 and 0. The discount factor controls how an agent regards rewards. The low values of the discount factor encourage myopic behavior where an agent will aim to maximize short-term rewards.

In contrast, high discount factor values cause agents to be more forward-looking and maximize rewards over a longer time frame. The expected value is the average over the whole thing, so that's the optimal policy that the agent can learn. Therefore, the value function means the expected return shows how good was the action for that given state. So, the expected return is equal to the value function. Hence, the optimal policy is simply the maximum value of that value function for that particular policy. The aims are to train the NN to perform the best optimal policy, and it is the task for the RL.

Figure 2. Component of RL.
2.3. Deep Q-Network

The DQN demonstrates how a Convolutional Neural Network (CNN) can learn successful control policies from just raw data. The network was trained end-to-end and was not provided with any specific information. In DQN-Network, the action is as an input to the NN. It consists of 3 convolutional layers and flattens layer to flatten the output of these layers. It then passes through Fully Connected layers, while the last layer is each node associated with the action. The outcome is a Q value associated with each action, and select the optimal action as the maximum of these Q values. Eventually, given enough experiences that train the NN, the NN will output Q-values associated with all actions given that state. So, it essentially learned the pattern of the possible Q values should be for the experiences that it has gone through for all given actions, and it will select the maximum Q-value, and it will always be the right action. DQN Architecture Network is demonstrated in figure 3.

![Figure 3. DQN architecture network.](image)

2.4. DQN problem formulation

2.4.1. Q Function. Q function is the same as the value function \( V \), except it takes in state and action pair:

\[
Q(s, a) = E[\sum_{t=0}^{T} \gamma^t r_t | s, a]
\]  
(4)

The optimal policy is now:

\[
\pi^* = \arg \max_Q (Q^*(s, a))
\]  
(5)

All the value functions obey particular equations, called the Bellman equations. The Bellman equivalent for the Q function:

\[
Q^*(s, a) = E[\pi \{ r_t + \gamma^t \max_a Q^*(S_{t+1}, a_{t+1}) \} | s, a]
\]  
(6)

Q-value function or also known as the action-value function, takes in state and action pair. It also can be defined as the expected return starting at some states but then taking an arbitrary action. Still, after it has achieved some experiences and learn from those experiences, the NN has some policy even though that policy is not strong. So, the Q-value function/action-value function then will be used in the DQN, and this essentially can be a substitute for the state value function in the same way. Hence, the optimal policy function is just a maximum of the optimal action-value function, similar to the optimal policy function being the maximum state value function.

2.4.2. DQN Loss Function. The Bellman Q value is the \( Q^\text{target} \) value. The output of the model is \( Q_\text{predicted} \) value. Loss is simply the Mean Square Error value of the two:

\[
L(w) = E[(Q^\text{target} - Q_\text{predicted}(s, a; w))^2]
\]  
(7)
The value function obeys the Bellman rule, which is a recursive rule which states that the value of a current state is the reward that gets for that particular state as well as the expected reward for the successor state, so the expected reward for the next state. According to the Bellman equivalent of the action-value function also, the optimal q value or the optimal value for the current state given some action is the reward achieve for that particular time step for that specific state plus the expected value or expected return achieve for the next state and the possible action the agent will take. So, the development equation and basic idea in RL are to estimate this optimal Q value by using this Bellman as an iterative update.

2.5. Policy learning algorithm
The policy learning algorithm for collision avoidance applying the DQN algorithm in RL.

Algorithm 1: Collision Avoidance Policy Learning Algorithm

**Input:** Action = A {Move forward, Turn Right, Turn Right}
State = X {1 \ldots \ldots \ldots N_s}
Output = Q (X, A) optimal State and Action
Let \( \gamma \) [0,1] - Discount factor,
Let \( \alpha \) {0,1} – Learning rate = 0.1
Let R – Reward
Initialize of replay memory D to capacity N
Initialize of action-value function network Q with random weights w
Initialize \( S = \) Random State, \( A = \) Random Action, \( R = \) Arbitrary
\( Q: S*A = R \)

**Output:** trained w; w - weight

For epi_idx = 1 to max_episodes do
  Randomly initialize the current state \( S_t \);
  While S is not terminal, do
    With probability S randomly select an action A with a certain probability;
    Otherwise, select action A corresponding to the maximum value of Q through Q-network;
    According to the current state \( S_t \), and selected action \( A_t \), control the vehicle and interact with the simulation environment in the CARLA;
    Calculate Policy (\( \pi(x) \)) - \( \arg \max (Q^*(s,a)) \)
    Action – Policy (state)
    If Collision == True
      reward – R (state, action)
      reward = N
    else
      reward – R (state, action)
      update (reward)
    end if
    \( s' = T(s,a) \) //Receive new state
    \( Q (s', a) \)
    \( s = s' \)
  end while
Calculate the loss function
return
end For
2.6. Reward function
To enable the agent to learn the desired strategy, RL algorithms depend on reward functions. The agent is penalized if a collision occurs and gets the rewards for efficiency and safety actions. Here, we adopt an analysis of reward function value $R$ that takes different values using three criteria. In this paper, the shaped reward consists of the values obtained by considering the speed of the agent in the environment. The various evaluations are defined in Table 1.

| Cases | Conditions |
|-------|------------|
| 1     | +1 for each frame driving > 50 KM/H  
      | -1 for each frame driving < 50 KM/H  
      | -200 for a collision and episode is over |
| 2     | +0.005(1/200) for each frame driving > 50 KM/H  
      | -0.005(-1/200) for each frame driving < 50 KM/H  
      | -1(-200/200) for a collision and episode is over |
| 3     | +1 for each frame driving > 50 KM/H  
      | -1 for each frame driving < 50 KM/H  
      | -1 for a collision and episode is over |

2.7. Traffic scenario
In a two-lane road scenario, as shown in Figure 4, the ego vehicle/agent (yellow car) is run over into the blocking vehicle (red car) at a different speed. The blocking vehicle moves at a low speed compared to the agent, which may collide if the agent maintains at the same speed on the same lane. The ego vehicle in the traffic has two options, either make a lane change or keep the current lanes. If it decides to change lanes, there may be two possible outcomes. If it changes lanes, it may rear collide with the approaching vehicle from another road lane or successfully merge in the new lane and run at its best speed. The blocking vehicle is assumed not to brake abruptly. If the agent chooses to keep the current lane, it has to perform a braking process to slow down the vehicle to match the blocking vehicle's speed. Therefore, the autonomous agent should adapt to different driving scenarios and make a quick and accurate decision.

In this study, the agent is controlled by an RL-base intelligence. It learns how to drive, including longitudinal speed control and lane-changing strategy, from interacting with the environment vehicles. There are also included some assumptions for the agent vehicle to carry out the path planning decision making of collision avoidance as follows:

- The ego vehicle is equipped with sensing capabilities and can measure the relative distances to the blocking vehicle.
- There is no communication between any vehicles.
- Each agent makes its driving decisions according to the driver’s decision-making model.
3. Performance Evaluation

The performance evaluation is evaluated using three criteria which are accuracy, loss, and reward average. The accuracy is used to evaluate each episode's performance whether the autonomous driving behavior is completed the task on the road segment. Successful cases, which are indicated by the accuracy percentages, show no collision; meanwhile, the failure cases show that the agent's accidents occur. Therefore, the excellent loss value is supposed to keep the value of 0 and below to get the model's best performance. For the reward average, the network's promising performance indicates that the reward average will keep increase and gradually stable and optimal after the agent gets enough experience in the environment. For this study, the model is trained for 10000 episodes.

To carry out the study of the multiple reward function performance first, we set the penalty evaluation value to -200, which is 200 more times than the reward evaluation. A penalty value larger than the reward value is suppressed. The penalty value is set to \( \geq 200 \) times higher because it is the amount to make sure that the agent will be more efficient in the learning process when it has given the higher punishment for the mistake/collision from the action that it takes.

3.1. Analysis performance: case 1.

![Figure 4. Two-lane traffic scenario.](image-url)
Figure 5. Analysis performance for case 1 (a) Accuracy, (b) Loss, and (c) Reward average.

Figure 5 shows the analysis performance for case 1. Figure 5(a) shows the accuracy performance of the network. The line graph of accuracy shows that it fluctuates, which means that the learning process of the model is not satisfied/reach the aim for the designated task. While figure 5(b) shows the loss value, which is there are the specific time that the performance is exploding which is at 1121 to 1534 and 6280 to 7525 episodes. Next, as shown in Figure 5(c), the average reward obtained is fluctuated in the simulation process, which means that the model is not stable in performing the task. By setting the penalty value range higher than the reward value, we found that this was likely exploding the Q values, which also seemingly, and now obviously, due to the monumental size of the collision penalty in comparison to all else. Also, possibly the bounds being out of range. Therefore, from the analysis, it cannot perform good collision avoidance.

Then, as the Q-value is exploding next, we set the penalty value in the form of a lower integer but still in the same range value as in case 1. So, the value of the reward and penalty are divide by 200 to
get the lower integer but the same range value in case 1. Thus, the penalty value is -1, and the reward value is set to +0.005.

3.2. Analysis performance: case 2. The analysis performance case 2 shows that in figure 6(a), we can see that at first, the accuracy is exploded at the beginning to learn from the process. It became stable with a high accuracy value; however, at episode 3814, it becomes fluctuate until the end of the training. Meanwhile, for the loss graph, as shown in figure 6(b), it explodes at the 7177 episode, which is supposedly it will explode at the early learning to learn the experience and gradually become constant at 0 value. Meanwhile, figure 6(c) show that the reward average improved pretty consistently. At the moment, it has slipped, but not for necessarily longer than it historically did a few times. Overall, the model did improve compared to case 1.
3.3. Analysis performance: case 3. Next, in Case 3, the penalty value is slightly increased than the reward. Which is there is no much gap of values ranges between penalty and reward. The agent will get the penalty (-1) if it is driving speed is <50 KM/H, which is in decelerate state, and (-2) if during the decelerations it collides with the obstacle. Analysis performance in case 3 shows that the model gives more stable performances. Figure 7(a) shows the fluctuate graph until episodes 789 become much more constant with high accuracy (>60%). From 0-789 episodes, the agent is still learning from the experience it gets in that environment. After 789 episodes, it is learning to avoid the collision, and the efficiency of the model is increased. As the highest percentage accuracy is achieved, the highest chance of the agent to complete the task assign by the designer while avoiding all of the obstacles. In terms of loss, as indicated in figure 7(b), the loss graph, which is at the beginning is exploded, and after that, it became constant at 0. This means that during the explode period, the agent is still getting the learning process experience then, after that, it can perform well, which is a constant 0 of the error value. Next, figure 7(c) shows the promising result of the average reward value as the line keeps increasing and gradually stable and optimal.
Figure 7. Analysis performance for case 3 (a) Accuracy, (b) Loss, and (c) Reward average.

The average accuracy of the three cases is demonstrated in Table 2. Setting the parameter of reward function for the DQN Algorithm, from the analysis, shows that there is a disadvantage in setting the penalty value higher than the reward value as the Q-value may explode. It is necessary to give a slightly different value between the reward and penalty as the model can perform/learn much better.

Table 2. Average accuracy.

| Cases | Conditions |
|-------|------------|
| 1     | 70%        |
| 2     | 85%        |
| 3     | 94%        |
4. Conclusion

The slow convergences and the urge to slip into the optimal local solution are improvements in RL. In this paper, the reward function performance for collision avoidance applying the DQN algorithm in RL has been studied. Significantly, the study provided the three cases for selecting the reward function model. We use the time-weighted reward function, which is speed, with the 50kmH threshold. Setting the reward function parameter from the study shows that there are advantages to set the penalty value higher than the reward value as the Q-value may explode. It is may due to the monumental size of the collision in comparison to all else. Therefore, by adopting different types of analysis using various appropriate reward functions, we were able to obtain the higher positive rewards and the optimum Q-values.

Acknowledgement

This work was supported in part by the CRG 8.1: AI Predictive Model for HVAC Operation under Grant Q.K130000.2443.04G73, and CRG 8: Development Smart Embedded-Analytics Sensors, Cloud-Based Management Systems for Heating, Ventilation, and Air Conditioning (HVAC) System under Grant Q.K130000.2443.04G64 and in part by the Malaysia Ministry of Education under Fundamental Research Grant Scheme FRGS/1/2019/TK08/UITM/03/1 File No. 600-IRMI/FRGS 5/3 (466/2019)

References

[1] Manju M S 2011 An analysis of Q-learning algorithms with strategies of reward function Int. J. Comput. Sci. Eng. 3 814–20
[2] Hu Z, Wan K, Gao X and Zhai Y 2019 A dynamic adjusting reward function method for deep reinforcement learning with adjustable parameters Math. Probl. Eng. 1 1–10
[3] Marashi M, Khalilian A and Shiri M E 2012 Automatic reward shaping in reinforcement learning using graph analysis 2012 2nd Int. eConf. Comput. Knowl. Eng. ICCKE 2012 (Mashhad, Iran) 111–6
[4] Matignon L, Laurent G J and Le Fort-Piat N 2006 Reward function and initial values: Better choices for accelerated goal-directed reinforcement learning Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics) 840–9
[5] Shaping P, Initialization Q and Wiewiora E 2003 Potential-based shaping and Q-Value initialization are equivalent J. Artif. Intell. Res. 19 205–8
[6] Ng, Andrew Y. Daishi Harada S R 2554 Policy invariance under reward transformations: Theory and application to reward shaping Int. Counc. Mach. Lubr. 3
[7] Hailu G and Sommer G 1999 On amount and quality of bias in reinforcement learning Proc. IEEE Int. Conf. Syst. Man Cybern. 2 1491–5
[8] Wei Y, Zhao M, Zhang F and Hu Y 2004 Research of a heuristic reward function for reinforcement learning algorithms Proc. World Congr. Intell. Control Autom. 3 2676–80
[9] Yan Q, Liu Q and Hu D 2010 A hierarchical reinforcement learning algorithm based on heuristic reward function IEEE Int. Conf. Adv. Comput. Control
[10] Mataric M J 1992 Reward functions for accelerated learning Int. Conf. Mach. Learn. 181–9
[11] Behnke S and Bennewitz M 2005 Learning to play soccer using imitative reinforcement ICRA 2005 Work. "Social Asp. Robot Program. through Demonstr.
[12] Mesot B, Sanchez E, Pena C A and Perez-Uribe A 2002 SOS++: Finding smart behaviors using learning and evolution 8th Int. Conf. Simul. Synth. Living Syst. Alife (Sydney, Australia) 264–73
[13] Wu Y H, Sun F Y, Chang Y Y and Lin S De 2018 ANS: Adaptive network scaling for deep rectifier reinforcement learning models arXiv:1809.02112v3 [cs.LG]
[14] Xiao L, Chen Y, Lin W S and Liu K J R 2012 Indirect reciprocity security game for large-scale wireless networks IEEE Trans. Inf. Forensics Secur. 7 1368–80