JOURNAL OF SOFT COMPUTING AND DATA MINING
e-ISSN: 2716-621X

JSCDM

Vol. 5 No. 1 (2024) 1-14
https://publisher.uthm.edu.my/ojs/index.php/jscdm

Comparative Analysis of Loss Functions in TD3 for Autonomous Parking

Ka Heng Chan[1], Aida Mustapha[1]*, Mohammed Ahmed Jubair[2]

[1] *Faculty of Applied Sciences and Technology,*
Universiti Tun Hussein Onn Malaysia, KM1 Jalan Panchor, 84600 Panchor, Johor, MALAYSIA
[2] *College of Information Technology,*
Imam Ja'afar Al-Sadiq University, 66002, Al-Muthanna, IRAQ

*Corresponding Author: aidam@uthm.edu.my
DOI: https://doi.org/10.30880/jscdm.2024.05.01.001

Article Info

Received: 1 December 2023
Accepted: 25 April 2024
Available online: 21 June 2024

Keywords

Twin delayed deep deterministic policy gradient algorithm, reinforcement learning, loss function, autonomous parking

Abstract

Autonomous parking is a revolutionary technology that has transformed the automotive industry with the rise of deep reinforcement learning, in particular, the Twin-Delayed Deep Deterministic Policy Gradient Algorithm (TD3). Nonetheless, the robustness of TD3 remains a significant challenge due to bias in Q-value estimates when determining how good an Action, *A*, taken at a particular state, *S*. To investigate this gap, this paper analyzes different loss functions in TD3 to better approximate the true Q-value, which is necessary for optimal decision making. Three loss functions are evaluated; Mean Squared Error (MSE), Mean Absolute Error (MAE) and Huber Loss via a simulation experiment for autonomous parking. The results showed that TD3 with Huber Loss has the highest convergence speed with the fastest Actor and Critic loss convergence. The Huber Loss function is found to be more robust and efficient than either loss function such MSE or MAE used in isolation, making it a suitable replacement for existing loss functions in the TD3 algorithm. In the future, TD3 with Huber Loss will be used as the base model to solve overestimation problem in TD3 when the estimated Q-values that represent the expected rewards of taking an action in a particular state, are higher than their true values.

1. Introduction

Autonomous parking is a technological advancement in the automotive industry that enables vehicles to park themselves without human intervention. This technology utilizes a combination of sensors, cameras, and algorithms to detect available parking spaces and maneuver the vehicle into the designated spot [15]. It not only reduces the time and effort required for parking but also minimizes the risk of accidents caused by human errors [10]. Autonomous parking systems are becoming increasingly common in modern vehicles and are expected to become a standard feature in the future. This technology represents a significant step towards fully autonomous driving and has the potential to revolutionize transportation and urban mobility.

In conventional autonomous parking systems, the vehicle is guided by manual rules and sensors that detect obstacles and help the car to navigate into the parking spot. However, these systems can be limited in their effectiveness, as they may struggle in complex or changing environments or with unusual parking configurations [7]. As opposed to the rule-based systems in autonomous parking, Reinforcement Learning (RL), one of the Machine Learning (ML) algorithm, offers a more flexible approach by allowing the autonomous systems to learn through trials and errors on how to navigate and park in a wide variety of environments [4]. With RL, the

autonomous parking systems can receive feedback on its actions and adjust its behaviour to optimize for success [13].

Machine learning is a constantly evolving field that aims to mimic human intelligence by allowing algorithm to finish some task [8]. It is a set of computational techniques that enable algorithms to improve their performance or make accurate predictions based on experience, which can take the form of past data gathered and made available for analysis in digital form. These data can be in the form of labeled training sets, unlabeled training sets or other information that is acquired through interactions with the environment [3, 19]. Because machine learning is data-driven and very dependent to dataset, therefore, the quality of dataset affect the performance of the algorithms. There are three types of ML algorithms, which are supervised learning, unsupervised learning and reinforcement learning. Supervised learning is mostly used in classification and regression tasks, which relies heavily on labeled data. Unsupervised learning is commonly used in clustering in the absence of labeled data. The third type of ML algorithm is the Reinforcement Learning (RL), which is independent from the past dataset. This algorithm trains itself by interacting with the environment and learn from mistakes based on trials and errors. Reinforcement learning is mostly used in tasks that have to achieve optimal solutions such as in gaming and robotics.

A Deep Reinforcement Learning (DRL) combines the conventional RL algorithms with deep neural networks in Deep Learning [18]. This family of learning algorithm has gained popularity to a wide range of application, including robotics [21], game playing [13], natural language processing [13], recommendation systems [6], tax collection estimation [22], as well as autonomous systems [2, 7, 11, 13, 18, 21]. The use of deep neural networks allow agents to learn complex patterns and representations from high-dimensional inputs, which can enable high complexity decision-making in complex environments. One particular RL algorithm is the Twin Delayed Deep Deterministic Policy Gradient (TD3), which is commonly used to train agents in complex environments, such as in robotics, games, and autonomous vehicles [1]. It builds upon the Deep Deterministic Policy Gradient (DDPG) algorithm, but with several modifications and improvements that address some of the key challenges in training deep reinforcement learning models, such as overestimation bias and exploration-exploitation trade off [9].

The TD3 algorithm uses two critics to estimate the value of an action taken by the agent, which helps to reduce the variance and improve the stability of the learning process. It also incorporates a target policy smoothing technique, which adds noise to the policy during training, to encourage exploration and prevent the agent from getting stuck in local optima. TD3 has demonstrated impressive results in various challenging tasks, such as robot locomotion, object manipulation, and game playing; surpassing the performance of previous RL algorithms.

Given this capabilities, TD3 is a promising algorithm for autonomous parking, which is a challenging problem in setting the vehicle to navigate through complex and dynamic environments such as crowded parking lots and tight spaces, while avoiding collisions with other vehicles and obstacles. By learning from its own experience, TD3 can effectively navigate the vehicle to the desired parking spot with minimal human intervention with its ability to handle continuous action and observation spaces. Additionally, its robustness and stability make it capable of handling the noise and uncertainty present in real-world parking scenarios. However, one of the key challenges in DRL is overestimation, which often leads to sub-optimal or even incorrect policies. This occurs when the estimated Q-values, which represent the expected rewards of taking an action in a particular state, are higher than their true values [16]. Overestimation can be particularly problematic in high-dimensional and noisy environments, where the noise can cause the algorithm to overvalue certain actions. Similarly, overestimation poses a problem to TD3. In TD3, overestimation can occur due to the use of two critics, which can produce biased Q-value estimates.

In TD3, a loss function is used to compare the Q-value estimates and the Q-target, which represent the expected future rewards of taking a certain action in a particular state. With the correct loss function, a TD3 agent would be able to learn and maximize the reward gained during training. The literature shows that different loss function may achieve higher performance in many deep reinforcement learning algorithm and that a loss function is a crucial component that determines an algorithm's performance [25]. One common loss function is the Mean Squared Error (MSE) that is often used to train agents to estimate Q-values, In TD3, MSE is used to update policy network and minimize the Q-value. In this paper, the effectiveness of MSE used in a standard TD3 algorithm is compared against two potential loss functions, which are Mean Absolute Error (MAE) and Huber Loss

The remaining of this paper is structured as follows. Section 2 presents work related to TD3 algorithms and the loss functions. Section 3 presents the simulation methodology. Section 4 presents and discusses the results. Finally, Section 5 concludes the paper with some direction for future work.

J. of Soft Computing and Data Mining Vol. 5 No. 1 (2024) p. 1-14

2. Literature Review

The Twin Delayed Deep Deterministic Policy Gradient (TD3) is a Deep Reinforcement Learning (DRL) algorithm that was first introduced by [9] in 2018. TD3 is an improvement over the original Deep Deterministic Policy Gradient (DDPG) algorithm, which was introduced by [14] in 2015. DDPG was the pioneering DRL algorithm continuous control problems, as it allows training of policies for agents to take continuous actions in continuous state spaces. However, DDPG was found to suffer from overestimation of Q-values, which led to sub-optimal policies. TD3 addressed this problem by proposing two modifications to the DDPG algorithm. First, TD3 uses a pair of critics, rather than a single critic, to estimate the Q-values, resulting in lower overestimation. The term "Twin" means TD3 learns two Q-functions and two Q-values to form the target. The term "Delayed" means TD3 updates its target network less frequently than the Q-function. Since there are two different outputs of Q-values, the smaller Q-value will be used for the target. Second, TD3 adds noise to the target policy during training, which helps to prevent the policy from getting stuck in local optima. TD3 has been shown to outperform DDPG and other state-of-the-art algorithms on a variety of continuous control tasks, including autonomous system.

Since its introduction, TD3 has become a widely used algorithm in the field of deep reinforcement learning for continuous control problems. However, overestimation remains a challenge in TD3, particularly in high-dimensional and noisy environments such as in autonomous parking systems. Overestimation occurs when the estimated Q-values, which represent the expected rewards of taking an action in a particular state, are higher than their true values, hence leading to sub-optimal or even erroneous policies. To investigate this problem within the context of autonomous parking, this paper attempts to examine various loss functions in TD3 to estimate the true Q-value more accurately, which is crucial for making optimal decisions.

At its core, a loss function evaluates how an algorithm models the dataset. The higher the output of a loss function, the lower performance a model is. Loss functions in deep reinforcement learning play a crucial role in training the agents to learn optimal policies in complex environments. Loss functions determine how much the agent's predicted Q-values deviate from the true Q-values and guide the learning process towards convergence to an optimal policy. One popular loss function in deep reinforcement learning is the Mean Squared Error (MSE), which measures the squared difference between the predicted Q-values and the target Q-values. When the difference between actual and predicted value is large, the value of MSE will be exponentially increased, hence causing large loss. Although MSE converges faster and able to achieve a considerably high accuracy, MSE is very sensitive to outliers and noise [25]. This is because the outlier will cause a very high MSE value, hence the training model will tend to focus more on outliers rather than adjusting the parameter to reduce MSE. This will result a lower precision and accuracy of the training model and will even reduce the overall performance.

The Mean Absolute Error (MAE) is a popular alternative loss function to the MSE across many machine learning tasks, including deep reinforcement learning [23]. MAE measures the absolute difference between the predicted value and the actual value. This difference is expressed as an absolute number, which represent the absolute error. Although MAE does not converge as fast as MSE, MAE is less sensitive to outliers, which can make it more robust in noisy environments. However, one potential drawback of MAE is that it can be less effective at distinguishing between small errors and large errors. This, in turn, makes it less effective at fine-tuning policies in some situations [24]. Since it is less smooth as MSE, the fixed gradient of MAE is not the exact value of loss value [25]. This can give a negative impact to the model training efficiency.

In reinforcement learning, MAE is still commonly used to train agents to estimate Q-values, which represent the expected future rewards of taking an Action, A, taken at a particular state, S. An alternative to MSE and MAE is the Huber Loss, which is a combination of MSE and MAE. Huber loss is designed to be less sensitive to outliers than MSE, while still being able to provide a smooth gradient near the minimum. It achieves this by using a hybrid loss function that combines the advantages of both MSE and MAE [25]. The parameter δ in Huber Loss function marks a boundary to differentiate the samples. Sample within the boundary make use of MSE, while sample outside of the boundary use the MAE. This approach decreases the impact of outlier and preventing the overfitting of the model. Although δ is advantageous, it makes the Huber Loss become a complex loss function that needs more iterations to train.

Several recent studies have explored the use of Huber Loss in reinforcement learning tasks. [17] focused on the cart-pole balancing problem and applied Q-learning and the Deep Q-Network (DQN). The research used MSE and Huber Loss for both DQN and Double DQN (DDQN) reinforcement learning models. Their results showed that the use of the Huber Loss function is more effective in achieving fast convergence and accuracy in comparison to the MSE loss function in cart-pole balancing problem. Similar finding was reported by [5] who tested on the Cart Pole, Acrobot, Lunar Lander, and MountainCar problems, which are all classic problems in reinforcement learning. These problems require agents to learn effective policies for making decisions based on a set of observations and rewards.

In transportation domain, [20] used the Huber Loss with a deep reinforcement learning algorithm to solve the eco-approach and departure problem for minimizing fuel consumption near signalized intersections. Similarly, [26] replaced MSE with Huber Loss in simulating autonomous vehicle control with the Convolutional Neural Networks (CNN) to extract the information from image data and the Double Deep Q-Network (Double DQN) algorithm to train the agent to drive the car. Huber Loss was found to be able to increase training stability in solving both problems.

3. Materials and Methods

This paper analyzes the use of different loss functions in Twin-Delayed Deep Deterministic Policy Gradient Algorithm (TD3) for autonomous parking problem, which are the Mean Squared Error (MSE), Mean Absolute Error (MAE), and Huber Loss. The objective of this analysis is to identify the most suitable loss function that improves the performance of TD3 algorithm.

3.1 TD3 Algorithm

For TD3 algorithm, there are two components that the agent extract from the environment at each iteration. The first component is the information about the car's current condition, which includes the position on the *x* and *y*-axes, velocity, acceleration, direction of its front wheels and the sine and cosine of heading angle of the car agent. These five elements are used to find out the most optimal path for parking. The agent possesses the same level of control over the vehicle as any regular automated vehicle, with the ability to manipulate both its acceleration (a) and the direction of its front wheels (δ). The steering angle can vary within the range of -π/3 to π/3 radians, while the acceleration can be adjusted within the range of -5 to 5 m/s^2. This will enable the vehicle to navigate in both forward and backward directions.

The second component is the reward function. The reward function is an essential component of deep reinforcement learning because it determines the goal or objective that the agent should try to achieve. The reward function provides a numerical measure of how well the agent is performing its task and guides the agent to learn a policy that maximizes the cumulative rewards over time. The reward function maps the state and action pairs to a scalar reward value, which provides feedback to the agent on the quality of its decision-making. The reward function used in TD3 is shown in Equation 1.

$$r = -(X + 0.3Y + 0.02\cos\theta + 0.02\sin\theta)^{0.5} \quad (1)$$

Where *X* is the difference between *x*-coordinate of actual and desired position of the car, *Y* is the difference between *y*-coordinate of actual and desired position of the car, and θ is the difference between actual car heading angle and parking slot angle.

Based on Equation 1, *X* and *Y* are used to make sure that the car is able to be closer to the desired parking slot, while θ is used to make sure the car park not only in the right place but according to the angle of the parking slot. The further the distance between the car and the parking slot, the lower the reward gained. The higher the difference between the car heading angle and the parking slot angle, the lower the reward gained. The agent is rewarded if the car does not meet the terminal condition.

There are four terminal conditions that impact the reward, which include the agent colliding with an obstacle, distance travelled exceed 100 steps and the agent park to the desired parking slot successfully. To determine whether an episode is successful or not, the distance from the vehicle's location to the goal spot's location and difference between heading angle and parking slot angle are calculated at each time step. If any of these conditions hold true, the episode is concluded, and the agent is reset. The episode is considered successful if the reward r more than -3.3. Due to the vehicle's slow speed, the issue of sliding can be disregarded.

The neural network structure used for the training DRL is an important factor that greatly affects the overall performance of the model. In this experiment, the structure consists of three hidden layers, each with 512 neurons. These layers are shared between the policy network and the value network, allowing for efficient use of computational resources. One of the key aspects of any machine learning algorithm is the selection of appropriate hyperparameters. In this case, the hyperparameters used for the training configuration of the DRL algorithms are presented in Table 1.

J. of Soft Computing and Data Mining Vol. 5 No. 1 (2024) p. 1-14

Table 1 *Hyperparameters used for training*

Hyperparameter	Value
Learning rate	0.001
Replay buffer size	1000000
Discount factor	0.95
Batch size	1024

These hyperparameters play a critical role in determining the overall accuracy and efficiency of the model and are carefully chosen to ensure optimal performance.

3.2 TD3 with Mean Square Error (TD3+MSE)

To solve overestimation of TD3 algorithm, changing the loss function that calculating the Q-function is believed can reduce overestimation problem. In a standard TD3 algorithm, the Mean Squared Error (MSE) is used in calculate Q-function, which is the difference between the actual and the predicted Q-value. The objective of MSE in TD3 is to update policy network and minimize the Q-value. MSE evaluate the error by using average squared difference between observed and predicted values. If the predicted value has no difference with observed value, MSE will be 0. The value of MSE will increase when the difference rises. Equation 2 shows the formula of MSE and Figure 1 shows the plot of MSE loss,

$$MSE = \sum \frac{(y_i - \hat{y}_i)^2}{n} \tag{2}$$

Where y_i is the i^{th} observed value, $\hat{y}_i$ is the i^{th} predicted value and n is the number of data.

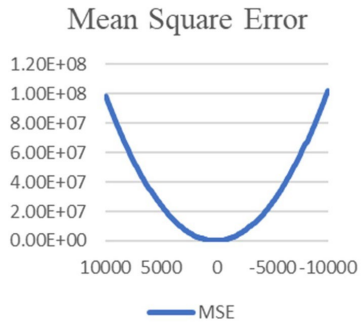

Fig. 1 *Plot of MSE loss*

A good estimator will have a low MSE. The function of MSE in TD3 algorithm is to update its critic network by minimizing the loss function. TD3 uses MSE to calculate the difference between the expected and predicted Q-value so that TD3 can identify how good an action taken. Figure 2 shows the pseudocode of TD3 with MSE. However, the presence of noise and approximation error in Q-value function can cause the maximum value to be greater than the true maximum.

Algorithm 1 TD3+MSE

Input: initial policy parameters θ, Q-function parameters $\phi_{tagr,1}$, ϕ_2, empty replay buffer D
Set target parameters equal to main parameters $\theta_{targ} \leftarrow \theta$, $\phi_{targ,1} \leftarrow \phi_1$, $\phi_{targ,2} \leftarrow \phi_2$
repeat
  Observe state s and select action $a = clip(\mu_\theta(s) + \epsilon, a_{Low}, a_{High}), where \epsilon \sim N$
  Execute a in the environment
  Observe next state s', reward r and send signal d to indicate if s' is terminal
  Store (s, a, r, s', d) in replay buffer D. If s' is terminal, reset environment.
  if it's time to update **then**
    for j in range (however many updates) **do**
      Randomly sample a batch of transitions B={(s, a, r, s', d)} from D
      Compute target actions:

$$a'(s') = clip(\mu_{\theta_{targ}}(s') + clip(\epsilon, -c, c), a_{Low}, a_{High}),$$

      Compute targets:

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{targ,i}(s', a'(s'))$$

      Update Q-functions by one step of gradient descent:

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_{\phi_i}(s, a) - y(r, s', d))^2$$

      if j mod $policy\_decay$=0 **then**
        Update policy by one step of gradient ascent:

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} (Q_{\phi_1} s, \mu_\theta(s))$$

        Update target networks:

$$\phi_{targ,i} \leftarrow \rho\phi_{targ,i} + (1 - \rho)\phi_i \theta_{targ} \leftarrow \rho\theta_{targ} + (1 - \rho)\theta$$

      end if
    end for
  end if
until *converge*

Fig. 2 *Pseudocode of TD3 with MSE*

3.3 TD3 with Mean Absolute Error (TD3+MAE)

Although the Mean Absolute Error (MAE) is a common metric to evaluate the error between predicted and target values, MAE does not weigh error differently. MAE is calculated as the average of the differences between expected and predicted values. MAE increase linearly when the difference between actual and desired value increase. Equation 3 shows the equation of MAE and Figure 3 shows the plot of MAE loss,

$$MAE = \sum \frac{|y_i - \hat{y}_i|}{n} \tag{3}$$

Where y_i is the i^{th} observed value, $\hat{y}_i$ is the i^{th} predicted value and n is the number of data.

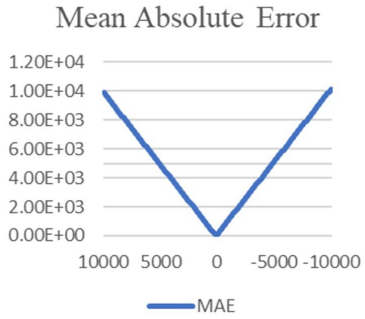

Fig. 3 *Plot of MAE loss*

Next, Figure 4 shows the pseudocode of TD3 with MAE loss function.

J. of Soft Computing and Data Mining Vol. 5 No. 1 (2024) p. 1-14

Algorithm 2 TD3+MAE

Input: initial policy parameters θ, Q-function parameters $\phi_{tagr,1}$, ϕ_2, empty replay buffer D
Set target parameters equal to main parameters $\theta_{targ} \leftarrow \theta$, $\phi_{targ,1} \leftarrow \phi_1$, $\phi_{targ,2} \leftarrow \phi_2$
repeat
Observe state s and select action $a = clip(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim N$
Execute a in the environment
Observe next state s', reward r, and send signal d to indicate if s' is terminal
Store (s, a, r, s', d) in replay buffer D
If s' is terminal, reset environment state.
if it's time to update **then**
for j in range **do**
Randomly sample a batch of transitions, B={(s, a, r, s', d)} from D
Compute target actions:

$$a'(s') = clip(\mu_{\theta_{targ}}(s') + clip(\epsilon, -c, c), a_{Low}, a_{High}),$$

Compute targets:

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{targ,i}(s', a'(s'))$$

Update Q-functions by one step of gradient descent:

$$\nabla_{\phi_i} \sum_{(s,a,r,s',d) \in B} MAE(Q_{\phi_i}(s, a), y(r, s', d))$$

if j mod $policy\_decay$=0 **then**
Update policy by one step of gradient ascent:

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} (Q_{\phi_1} s, \mu_\theta(s))$$

Update target networks:

$$\phi_{targ,i} \leftarrow \rho \phi_{targ,i} + (1 - \rho)\phi_i$$

$$\theta_{targ} \leftarrow \rho \theta_{targ} + (1 - \rho)\theta$$

end if
end for
end if
until $converge$

Fig. 4 *Pseudocode of TD3 with MAE*

3.4 TD3 with Huber Loss (TD3+Huber Loss)

Huber Loss is used to compute the error between predicted and expected values. Huber Loss combines advantages of both MAE and MSE, hence it is less sensitive to outliers and offer a smoother result as compared to MAE. Equation 4 shows the equation of Huber Loss and Figure 5 shows the plot of Huber Loss,

$$HuberLoss = \begin{cases} 0.5(y_i - \hat{y}_i)^2 & \text{for} |y_i - \hat{y}_i| < 1 \\ |y_i - \hat{y}_i| - 0.5 & \text{otherwise} \end{cases} \quad (4)$$

where y_i is the i^{th} observed value, $\hat{y}_i$ is the i^{th} predicted value and n is the number of data.

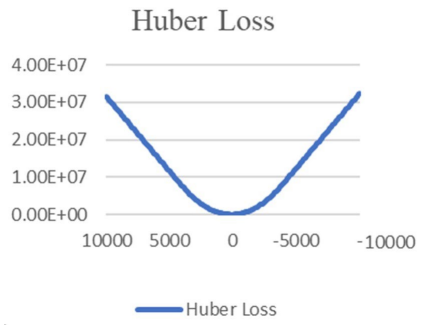

Fig. 5 *Plot of Huber Loss*

Next, Figure 6 shows the pseudocode of TD3 with Huber Loss.

Algorithm 3 TD3+Huber Loss

Input: Initial policy parameters θ, Q-function parameters $\phi_{tagr,1}$, ϕ_2, empty replay buffer D
Set target parameters equal to main parameters $\theta_{targ} \leftarrow \theta$, $\phi_{targ,1} \leftarrow \phi_1$, $\phi_{targ,2} \leftarrow \phi_2$
repeat
  Observe state s and select action $a = clip(\mu_\theta(s) + \epsilon, a_{Low}, a_{High}), where \epsilon \sim N$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate if s' is terminal
  Store (s, a, r, s', d) in replay buffer D
  If s' is terminal, reset environment state
  if it's time to update **then**
    for j in range **do**
      Randomly sample a batch of transitions
      $B=\{(s, a, r, s', d)\}$ from D
      Compute target actions:

$$a'(s') = clip(\mu_{\theta_{targ}}(s') + clip(\epsilon, -c, c), a_{Low}, a_{High})$$

      Compute targets:

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{targ,i}(s', a'(s'))$$

      Update Q-functions by one step of gradient descent:

$$\nabla_{\phi_i} \sum_{(s,a,r,s',d) \in B} HuberLoss(Q_{\phi_i}(s, a), y(r, s', d))$$

      if j mod $policy\_decay$=0 **then**
        Update policy by one step of gradient ascent:

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} (Q_{\phi_1} s, \mu_\theta(s))$$

        Update target networks:

$$\phi_{targ,i} \leftarrow \rho \phi_{targ,i} + (1 - \rho)\phi_i$$

$$\theta_{targ} \leftarrow \rho \theta_{targ} + (1 - \rho)\theta$$

      end if
    end for
  end if
until $converge$

Fig. 6 *Pseudocode of TD3 with Huber Loss*

3.5 Simulation Setup

The experiment is conducted in the environment that simulated by autonomous driving simulation tool Highway-env that coded in Python [12], which is based on Open AI's Gym environment toolkit for reinforcement learning algorithms. OpenAI Gym is a Python library that provides a collection of environments for developing and comparing reinforcement learning algorithms. Library that used in this paper include pygame, stable baseline3 and tensorboard.

- Pygame provides the platform for creating 2D car and parking environment.
- Stable baseline3 is used for implementing TD3 algorithm.
- Tensorboard is used to visualize and analyze the training metrics such as reward, distance travelled, success rate, actor loss and critic loss.

Pygame and stable baseline3 offers a range of autonomous driving environments and allows for the creation of custom environments to suit specific requirements. The Kinematic Bicycle Model is used to accurately simulate car behavior, where every car is rendered according to its *x* and *y* coordinates, angle of its heading and the steering angle. In each time step, the next state of each vehicle is simulated using velocity and steering rate inputs while adopting to the physics of their bodies. In this project, we used a modified version of Highway-env's parking environment, which includes dynamic and configurable generation of parking slots of different parking spot locations.

In this setting, the algorithms are trained separately, with one algorithm being trained at a time. The training for each algorithm is performed for a fixed number of steps, and subsequently, the logs generated during the training are used to evaluate the algorithms. At each step, the data is recorded and then logged at the end of each episode. In order to avoid the scenario where the car gets trapped in a loop, which would halt

J. of Soft Computing and Data Mining Vol. 5 No. 1 (2024) p. 1-14

the training, the episode is terminated and reset after 100 steps, irrespective of the agent's current state. The vehicle's motion planning occurs entirely within the simulation, with the vehicle able to move forwards and backwards at a maximum speed of 40 m/s. It can turn its front wheels up to π/3 radians in each direction. The agent controls two parameters: the steering angle of the car's front wheels, and its acceleration a (forwards or backwards), and the car reacts accordingly.

3.6 Simulation Setup

In this paper, exploratory analysis of the data obtained from the simulation experiments was performed. The data relevant to the comparison between TD3+MSE, TD3+MAE and TD3+Huber Loss include the success rate, average reward gained, average distance travelled, actor loss, critic loss, converge episode and running time to identify the best and the most suitable loss function for TD3.

Success rate is an essential indicator that identify how good a deep reinforcement learning is. The higher the success rate, the better the performance of deep reinforcement learning. A faster increase in the success rate indicates a better algorithm and a quicker convergence speed. Equation 5 and Equation 6 shows the calculation of success and success rate.

$$\text{Success} = \begin{cases} 1 & \text{r>-3.2} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

$$\text{Success Rate} = \sum_{i=0}^{N} \frac{Success_i}{N} \tag{6}$$

Reward gained by an agent is a scalar value that the agent receives from the environment after taking an action, indicating how good a agent is doing its task. The higher the reward gained, the better the performance of deep reinforcement learning. Equation 7 shows the calculation of average reward gained.

$$\text{Average Reward Gained} = \sum_{i=0}^{N} \frac{r_i}{N} \tag{7}$$

Distance travelled by the car is also an important indicator since the car must park to the desired parking slot in shortest path.

4. Result and Discussion

The Twin-Delayed Deep Deterministic Policy Gradient Algorithm (TD3) is trained without usage of sample data because in deep reinforcement learning, the agents learn from mistakes. The simulation experiments in autonomous parking with TD3 produced the following data:

- Success rate
- Reward gained
- Distance travelled
- Actor loss
- Critic loss

The TD3 algorithm were trained for 100000 episodes and all data were logged at the end of episode. The same setting is used for evaluating the Mean Squared Error (MSE), Mean Absolute Error (MAE), and Huber Loss, respectively. For comparative analysis, the data were then visualized using the Tensorboard, a visualization toolkit TensorFlow.

Figure 7, Figure 8, Figure 9, Figure 10, and Figure 11 compare the success rate, average reward, average distance travelled, actor loss, and critic loss for TD3, respectively.

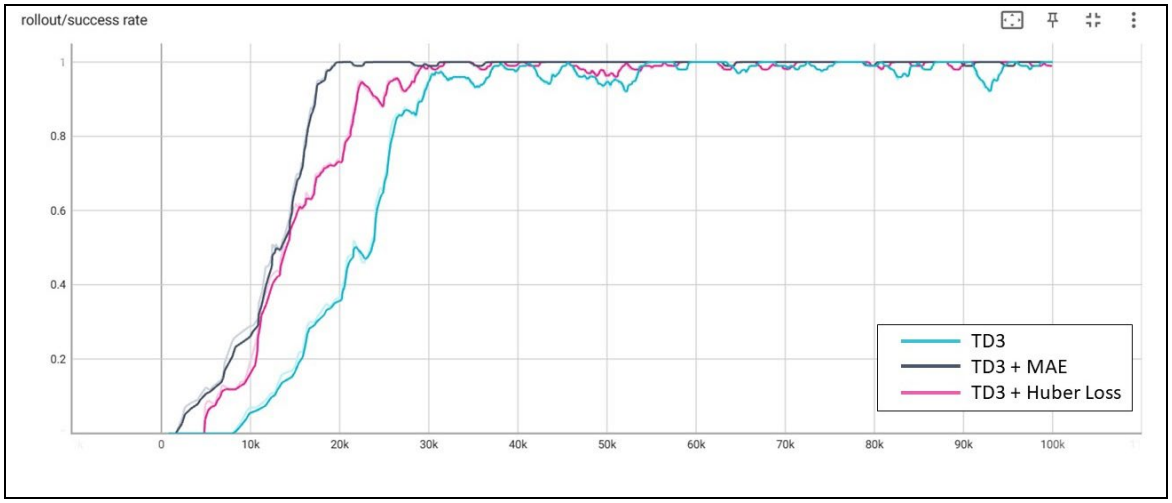

Fig. 7 *Success rate of TD3 with different loss function*

From Figure 7, the success rate of TD3+MAE started to increase at the episode 2400, TD3+MSE at episode 8652, and TD3+Huber Loss at episode 4904. TD3+MAE reached 100% success rate at episode 19504, TD3+MSE at episode 45612, and TD3+Huber Loss at episode 32123. From this figure, all three algorithms with different loss function achieved 100% success rate in training the autonomous parking agent. TD3+MAE has the highest speed in increasing the success rate, followed by TD3+Huber Loss and TD3+MSE.

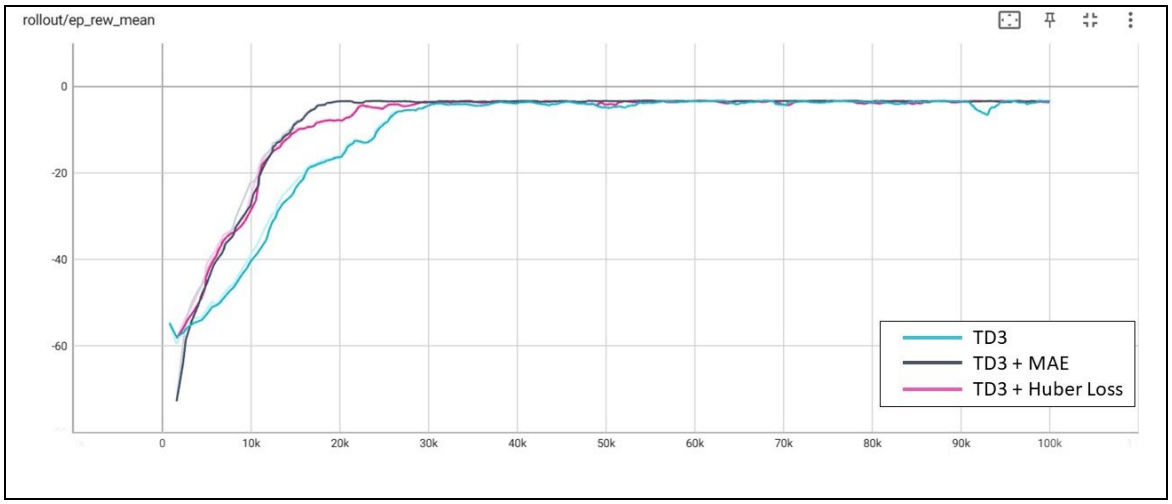

Fig. 8 *Average reward of TD3 with different loss function*

From Figure 8, TD3+MAE achieved the highest average reward gained of -3.37, while TD3+Huber Loss function achieved the lowest total reward gain of -3.65. Similarly, TD3+MAE achieved the lowest average distance travelled of 13.0, while the TD3+Huber Loss achieved the highest average distance travelled of 14.5. It can be seen from this figure that the convergence speed of average reward gained for TD3+MAE and TD3+Huber Loss are almost the same. For TD3+MSE, convergence speed of average reward gained are lower than TD3+MAE and TD3+Huber Loss.

J. of Soft Computing and Data Mining Vol. 5 No. 1 (2024) p. 1-14

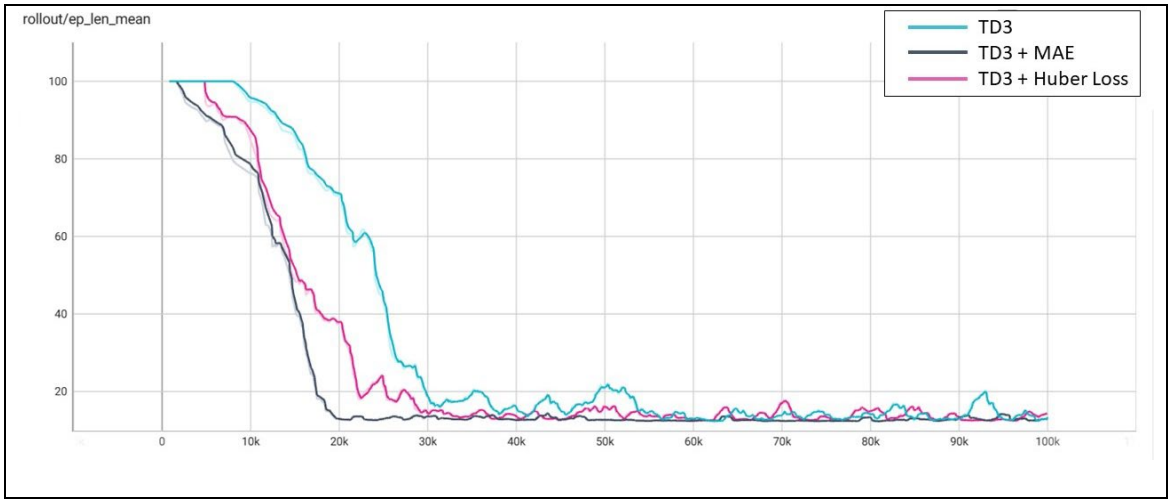

Fig. 9 *Average distance travelled of TD3 with different loss function*

Based on the results of average distance travelled in Figure 9, TD3+MAE has shown the ability to rapidly identify better path for the car to park, followed by TD3+Huber Loss and TD3+MSE. These shows that outliers are the key factor that affect the performance of TD3 algorithm. In addition to the success rate, average rewards, and average distance travelled produced by the TD3 algorithm, actor loss and critic loss were also considered in this study. The actor loss represents the difference between the predicted and actual actions taken by the agent, while the critic loss represents the difference between the predicted and actual values of the state-action pairs.

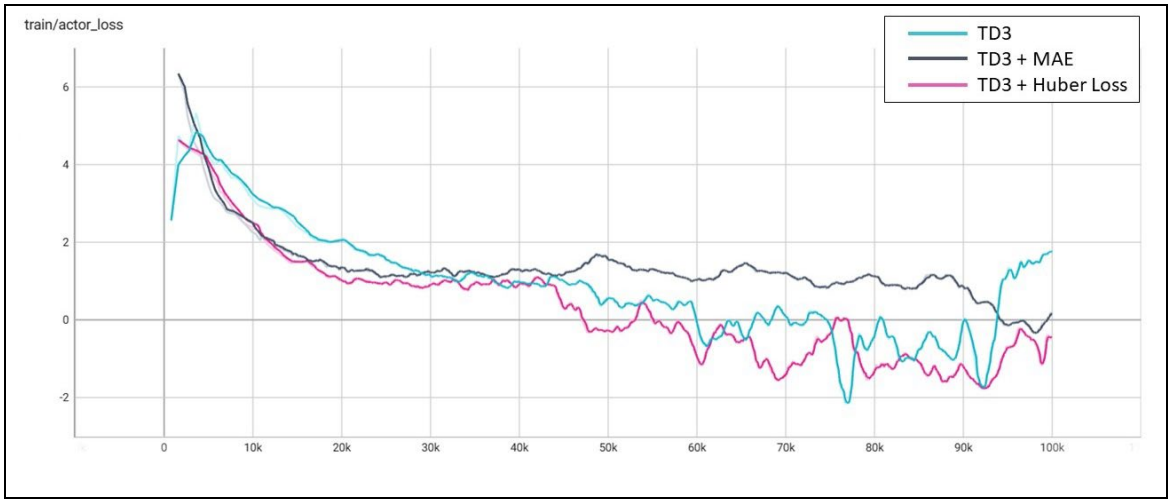

Fig. 10 *Actor loss of TD3 with different loss function*

The TD3+Huber Loss achieved the lowest actor loss of -0.465 and the lowest critic loss of 0.00377. On the other hand, the TD3+MSE achieved the highest actor loss of 1.75, while the TD3+MAE loss function achieved the highest critic loss of 0.0481. From Figure 10, it can be seen that the convergence speed of actor loss of TD3+Huber Loss is higher than TD3+MAE and TD3+MSE. Meanwhile, from Figure 11, the critic loss of TD3+Huber Loss has constantly low value compared to TD3+MAE and TD3+MSE. The convergence speed of critic loss for TD3+Huber Loss is higher as compared to TD3+MAE and TD3+MSE.

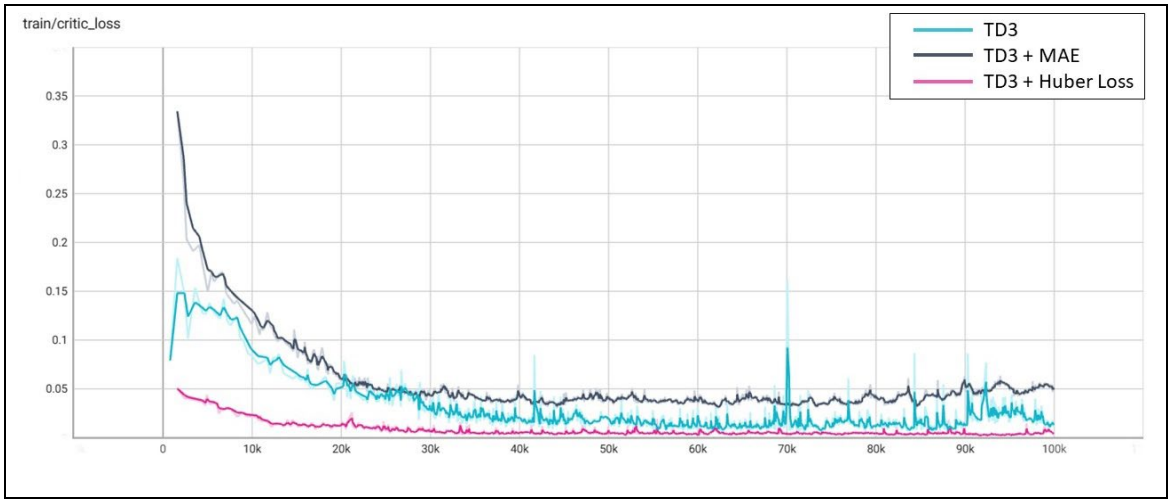

Fig. 11 *Critic loss of TD3 with different loss function*

Based on the simulated result, TD3+MAE converged in episode 19504, while TD3+MSE and TD3+Huber Loss converged in episode 45612 and episode 32123, respectively. The time used to finish 100000 training episode for TD3+MAE, TD3+MSE and TD3+Huber Loss was 4 hours 34 minutes, 4 hours 14 minutes, and 3 hours 46 minutes, respectively. TD3+Huber Loss has the highest convergence speed compared to TD3+MAE and TD3+MSE. This shows that replacement of MSE to Huber loss in TD3 algorithm fasten the learning speed. Moreover, as shown in Figure 10 and Figure 11, replacement of Huber loss help TD3 algorithm was able to reduce the overestimation of TD3 algorithm. The choice of loss function significantly impacts the training performance of the TD3 algorithm.

As a general observation, the Huber Loss has a high computational complexity due to the combination of the MAE and MSE components. However, this additional computational complexity is offset by the higher convergence rate and lower training time when compared to the other loss functions under study, which are MSE and MAE. In fact, the results shows that the Huber Loss has the lowest time consumption among the three loss functions.

5. Conclusion

This paper was set to analyze the effect of different loss functions in a Twin-Delayed Deep Deterministic Policy Gradient Algorithm (TD3) to better approximate the true Q-value in making optimal decision for autonomous parking. Three loss functions are evaluated; Mean Squared Error (MSE), Mean Absolute Error (MAE) and Huber Loss to determine the best loss function for the TD3 algorithm.

The results showed that TD3+Huber Loss had the highest convergence speed compared to TD3+MAE and TD3+MSE, and it helped to reduce the overestimation of the TD3 algorithm. TD3+MAE can rapidly identify better paths for the car to park, followed by TD3+Huber Loss and TD3+MAE. The convergence speed of actor loss and critic loss were also higher for TD3+Huber Loss compared to TD3+MAE and TD3+MSE. This shows that the Huber Loss combines the advantages of both the MAE and MSE loss functions, making it more robust and efficient than either loss function alone. Therefore, the replacement of MSE with Huber loss in TD3 algorithm can help to improve the learning speed and the overall performance of the algorithm.

In the future, this research will proceed in three directions. One is to investigate other loss functions such as the Quantile regression loss or the Wasserstein loss. Two is to explore different evaluation metrics that are specific for autonomous parking. Three is enhance the TD3 algorithm with different combination of hyperparameters, which can potentially affect the convergence speed and learning rate of the TD3 algorithm.

Acknowledgement

This research is funded by Universiti Tun Hussein Onn Malaysia under the Research Enhancement-Graduate Grant Vot Q078.

Conflict of Interest

Authors declare that there is no conflict of interests regarding the publication of the paper.

Author Contribution

*The authors confirm their contribution to the paper as follows. Chan contributed to the **design and implementation** of the research, to the **analysis of the results** and to the **writing of the manuscript**. Mustapha assisted in **analysis and editing** manuscripts. Jubair produced all figures from the **experiment and analysis**. All authors reviewed the results and approved the final version of the manuscript.*

References

[1] Abo Mosali, N., Shamsudin, S., Alfandi, O., Omar, R., and Al-fadhali, N. (2022). Twin delayed deep deterministic policy gradient-based target tracking for unmanned aerial vehicle with achievement rewarding and multistage training. IEEE Access, 10:1–1.
[2] Ahmed, S. A., Desa, H., T. Hussain, A.-S., and A. Taha, T. (2024). Implementation of deep neural networks learning on unmanned aerial vehicle based remote-sensing. IAES International Journal of Artificial Intelligence (IJ-AI), 13(1):941.
[3] Bensouda, N., El Fkihi, S., and Faizi, R. (2024). A novel ensemble model for detecting fake news. IAES International Journal of Artificial Intelligence (IJ-AI), 13(1):1160.
[4] Bhagat, S., Banerjee, H., Ho Tse, Z. T., and Ren, H. (2019). Deep reinforcement learning for soft, flexible robots: Brief review with impending challenges. Robotics, 8(1):4.
[5] Ceron, J. S. O. and Castro, P. S. (2021). Revisiting rainbow: Promoting more insightful and inclusive deep reinforcement learning research. In International Conference on Machine Learning, pages 1373–1383. PMLR.
[6] Chen, X., Yao, L., McAuley, J., Zhou, G., and Wang, X. (2023). Deep reinforcement learning in recommender systems: A survey and new perspectives. Knowledge-Based Systems, 264:110335.
[7] El Farnane, A., Youssefi, M. A., Mouhsen, A., and El Ihyaoui, A. (2024). Deep neural network for lateral control of self-driving cars in urban environment. IAES International Journal of Artificial Intelligence (IJAI), 13(1):1014.
[8] El Naqa, I. and Murphy, M. J. (2015). What is machine learning? Springer.
[9] Fujimoto, S., Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. In International conference on machine learning, pages 1587–1596. PMLR.
[10] Jhang, J.-H. and Lian, F.-L. (2020). An autonomous parking system of optimally integrating bidirectional rapidly-exploring random trees and parking-oriented model predictive control. IEEE Access, 8:163502-163523.
[11] Kiran, B. R., Sobh, I., Talpaert, V., Mannion, P., Al Sallab, A. A., Yogamani, S., and P´erez, P. (2021). Deep reinforcement learning for autonomous driving: A survey. IEEE Transactions on Intelligent Transportation Systems, 23(6):4909–4926.
[12] Leurent, E. (2018). An environment for autonomous driving decision-making. GitHub Repository.
[13] Li, Y. (2017). Deep reinforcement learning: An overview. arXiv preprint arXiv:1701.07274.
[14] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.
[15] Ma, Y., Liu, Y., Zhang, L., Cao, Y., Guo, S., and Li, H. (2021). Research review on parking space detection method. Symmetry, 13(1):128.
[16] Meng, L., Gorbet, R., and Kuli´c, D. (2021). The effect of multi-step methods on overestimation in deep reinforcement learning. In 2020 25th International Conference on Pattern Recognition (ICPR), pages 347–353. IEEE.
[17] Mishra, S. and Arora, A. (2023). A huber reward function-driven deep reinforcement learning solution for cart-pole balancing problem. Neural Computing and Applications, 35(23):16705–16722.
[18] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. nature, 518(7540):529–533.
[19] Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2018). Foundations of machine learning. MIT press.
[20] Mousa, S. R., Ishak, S., Mousa, R. M., Codjoe, J., and Elhenawy, M. (2020). Deep reinforcement learning agent with varying actions strategy for solving the eco-approach and departure problem at signalized intersections. Transportation research record, 2674(8):119–131.
[21] Nguyen, H. and La, H. (2019). Review of deep reinforcement learning for robot manipulation. In 2019 Third IEEE International Conference on Robotic Computing (IRC), pages 590–595. IEEE.
[22] Ourdani, N., Chrayah, M., and Aknin, N. (2024). Towards a new approach to maximize tax collection using machine learning algorithms. IAES International Journal of Artificial Intelligence (IJ-AI), 13(1):737.
[23] Qi, J., Du, J., Siniscalchi, S. M., Ma, X., and Lee, C.-H. (2020). On mean absolute error for deep neural network based vector-to-vector regression. IEEE Signal Processing Letters, 27:1485–1489.

[24] Schneider, P. and Xhafa, F. (2022). Anomaly Detection and Complex Event Processing Over IoT Data Streams: With Application to EHealth and Patient Data Monitoring. Academic Press.
[25] Wang, Q., Ma, Y., Zhao, K., and Tian, Y. (2020). A comprehensive survey of loss functions in machine learning. Annals of Data Science, pages 1–26.
[26] Yu, A., Palefsky-Smith, R., and Bedi, R. (2016). Deep reinforcement learning for simulated autonomous vehicle control. Course Project Reports: Winter, 2016.