Neural Computing and Applications (2023) 35:16705–16722
https://doi.org/10.1007/s00521-022-07606-6

S.I.: HUMAN-ALIGNED REINFORCEMENT LEARNING FOR AUTONOMOUS AGENTS AND ROBOTS

A Huber reward function-driven deep reinforcement learning solution for cart-pole balancing problem

Shaili Mishra[1] · **Anuja Arora**[1]

Received: 5 March 2022 / Accepted: 1 July 2022 / Published online: 30 July 2022

Abstract
Lots of learning tasks require experience learning based on activities performed in real scenarios which are affected by environmental factors. Therefore, real-time systems demand a model to learn from working experience—such as physical object properties-driven system models, trajectory prediction, and Atari games. This experience-driven learning model uses reinforcement learning which is considered as an important research topic and needs problem-specific reasoning model simulation. In this research paper, cart-pole balancing problem is selected as a problem where the system learns using Q-learning and Deep Q network reinforcement learning approaches. Pragmatic foundation of cart-pole problem and its solution with the help of Q learning and DQN reinforcement learning model are validated, and a comparison of achieved outcome in the form of accuracy and fast convergence is presented. An unexperienced Huber loss function is applied on cart-pole balancing problem, and results are in favor of Huber loss function in comparison with mean-squared error loss function. Hence, experimental study suggests the use of DQN with Huber loss reward function for fast learning and convergence of cart pole in balanced condition.

Keywords Reinforcement learning · Cart-pole problem · Q-learning · Deep Q learning · DQN · Double DQN

1 Introduction

The traditional method to steer the mechanical system problems is made up of analytical study of the system based on existing working principles and equations by manually adjusting the equation controlling parameters. These manually adjusting control parameters formed equation-driven approaches deal with enormous issues due to which solution capability is always questionable. The cart-pole system is an inherent and unstable mechanical system with nonlinear control dynamics [20, 21]. This under-actuated cart-pole mechanical system has lesser input parameters than the degree of freedom (DOF) which makes the cart-pole dynamics system a classic benchmark for designing, computing, testing and comparing different classical neural and evolutional models. Usually, under-actuated systems such as robot arm included no definite motion from the concepts of cart-pole dynamics, and these indefinite motions have lesser DOF from the point of view of kinematics [32, 33]. The control dynamics of cart-pole system involves the balancing mechanism such as rocket trajectory [1], human/robot walking [2, 3], robot arm movement [32, 33], and self-balancing dynamics system [4]. For solving such problems, some traditional approaches such as energy-based controllers [5], linear quadratic regulators (LQR) [6], fuzzy controller [7], and close-looping in structure [32, 33] are used, but computed result is not according to requirement in given time interval. Due to complex and variant nature of system, more enlightened controllers required for nonlinear, uncertain and time variant feature of mechanical system. For resolving such type of complex controller problem, reinforcement learning

✉ Anuja Arora
anuja.arora@jiit.ac.in

Shaili Mishra
19403028@mail.jiit.ac.in

[1] Department of Computer Science Engineering and Information Technology, Jaypee Institute of Information Technology, Noida, India

RL approaches are explored and presented tremendously efficient results in recent research studies. RL usually applied to research problems where output is not predefined and agents interact with environment by performing action corresponding to obtain maximum reward [8–10]. So, in case of classical cart-pole problem, reinforcement learning approach can generate optimal controller dynamics without previous knowledge of environment through the changing environment parameters [5, 6].

The evolving mathematical computational models suggest the use of reinforcement learning and optimization techniques to get a better automatized parameter controlling experimental study for the considered problem domain [11]. The research work performed in this same direction to validate the use of experience-guided reinforcement learning model to handle control parameters for engineering problems. Generally, engineering problems dynamically change state based on change in parametric factors and environment factors so huge amount of data is needed to train the model. The enormous training data are a viable solution to tackle all the successful, unsuccessful, and constraint-driven cases. Vast amount of data can encounter the engineering problem by building an effective supervised learning model, but supervised learning is not a feasible solution in case of environment controlled real-world problems. Environment control real-world problems do not have control over learning parameters/factors and because of that supervised learning model will not be feasible to handle unknown parameters impact while training. This frames the direction of work toward the use of reinforcement learning in combination with the concepts of deep neural network. Deep reinforcement learning is a deep learning epitome in RL which is the most trending AI models nowadays. Zhang et al. in their research work detailed various deep learning definitions and concluded that deep learning is a process to establish relation between two or more than two parameters, along with attainment of knowledge design between required function to proposed structure [34]. Deep RL is intersection of both deep neural network and reinforcement learning algorithms which is especially applicable and designed for high dimension and dynamic infinite state problem statements [8, 12].

The initial impetus was toward understanding and leaning the concepts of traditional RL and deep neural network. Furthermore, the applicability, implications, and efficiency of deep neural network along with reinforcement learning, i.e., deep reinforcement learning to handle complex real-time dynamic state changing problems, are explored. This class of learning algorithms addresses the problem of self-/automatic learning of optimal decision over time. Even, research issues those are initially handled by supervised or unsupervised learning to train a model based on standard static problem are converted to dynamically changing nature in real scenario. For example, in the case of cart-pole balancing problem, change in environmental factors can convert a static parameter-driven cart-pole balancing problem to dynamic problem due to involvement of added environmental factors, i.e., cart-pole balancing problem which will be affected by friction and gravity due to surface change (a change in environment condition) will not be a considerable factor for supervised/unsupervised learning solution. Some challenging applications of real-time control parameter problems examined through deep reinforcement learning are inverted pendulum modeling [13], lifelong robotic learning [14, 15], urban driving experience [16], UAV trajectory planning in wireless sensor network [17], and many more. Due to large population and urbanization, the intelligent transport system (ITS) has also applied decision-making reinforcement leaning techniques for better outcome. The combination of Deep RL with control-based systems as transportation, robot dynamic [18] and IOT gave satisfactory results. So, the single-agent and multi-agent RL environment shows better results for traffic signal control (TSC) than traditional approaches [7]. Like ITS system, some other control technologies support RL concepts as Real-Time Indoor Autonomous Vehicle Test Environment (RAVEN) which is also known as four-rotor inverted pendulum. This four-rotor inverted pendulum solved the multiple operations issues in same time interval with high efficiency. For the design of connecting rod path of the four-rotor, the reinforcement learning concepts applied for tracking and learning procedure and the outcomes depicts the satisfactory results also with high efficiency and less error [1].

Henceforth, reinforcement learning teaches the computer to learn from experience to take a decision made by an agent. This feature of RL solves very dynamic and complex real-life problems such as transportation problem, route finding problem, and so on. One of the RL applications is applying exploration and exploitation concepts on detecting optimum route between source and destination by using Deep Q Network (DQN) approach proposed by Wei et al. in 2020 [7, 19]. In this research work, researcher varies the epsilon value in epsilon-greedy algorithm and compute the optimum path [19].

In the studied literature for cart-pole problem, Nagendra et.al. compared the efficiency of several RL algorithms such as temporal difference, policy gradient, actor-critic approach, and value function approximation for pole balancing problem [5]. Some researchers decompose the process of RL and worked on varying factors of reinforcement learning such as define the agent, set finite states, reward function [20], and variable loss function, followed by policy gradient method [21]. Recent study performed by Mukherjee in 2021 compares the results effect of three different reward functions on Q-learning RL for solving cart-pole balancing problem [22] and shows the relation between performance and reward function. The first reward function reflects the earliest steady-state motion of pole, second reward function talks about lowest variability of pole for respective reward function, and third reward function is about nearest steady state [20]. Another 2021 study by Variengien et. al. is using Open AI Gym environment and neural cellular automata (Neural CA) framework on controlling procedure of cart-pole dynamics [4, 23]. This implements the concept of artificial neural network used for rules updating accordingly and the concept of deep neural network merge with Q-learning concept for model training [24]. The proposed system is combination of ANN for rule updating, integration of Neural Network with Q-learning (a reinforcement learning approach) gave extraordinary performance. The RL algorithms are totally based on agent, state, reward environment components. Q-learning is one of the promising approaches of RL which is based on Q-table that is pair of state and their corresponding actions. Q-learning approach with policy gradient factor improvises the performance of algorithm when applied on inverted pendulum traditional problem [10, 25].

The literature listed above shows that there are several studies existing that address the problem of cart-pole balancing. However, all these studies are valid considering reinforcement learning parameters in consideration. No detailed study has been introducing the concept of deep reinforcement learning for cart-pole balancing problem. Moving forward to this same direction, cart-pole balancing problem is taken as an application of inverted pendulum modeling to test the variety of controlling parameters of the study for fast convergence and to learn the consistently balance of pole on cart using deep reinforcement learning concept in this research paper. The research work is carried out to propose an efficient and robust deep reinforcement learning model for cart-pole balancing problem. Over and above a successful deep RL model could be used in a variety of real scenario simulation specifically varying applications of inverted pendulum. In summary, the research contributions of the presented work are:

- A clear picture of traditional reinforcement learning and deep reinforcement learning is acquainted.
- A novice research issue of environmental factors-based experience learning, i.e., reinforcement learning, is confronted for an inverted pendulum problem.
- A stable and robust deep reinforcement learning model is proposed for cart-pole balancing research issue.
- Practiced and evaluated a novel Huber loss reward function to measure fast (early) and stable convergence of cart-pole balancing position in comparison with mean-squared error loss function.

The rest of the manuscript is organized as follows: Section 2 overviews the cart-pole balancing problem according to the theoretical perspective. Foundation of computational techniques, i.e., learning models used to handle cart-pole balancing problem, is discussed in Sect. 3. Section 4 is about proposed methodology which is a deep reinforcement learning model. This section also discusses about all cart-pole parameters mapping with deep reinforcement learning concepts in detail for readers' point of view. Experimental setup and results with detailed discussions are presented in Sect. 5. Finally, concluding remarks are discussed in Sect. 6.

2 Theoretical foundation: cart-pole balancing problem

The well-known classics control problem is cart-pole balancing problem. In the mechanism of cart pole, a pendulum is joint with the cart through some pivot point from which pendulum can swing in one dimensional, i.e., horizontally with some initial constrained. The visual depiction is shown in Fig. 1, a horizontal force $F(t)$ implied on cart in x direction to control the motion of pendulum in upward direction. The main objective of cart pole is to prevent falling of pole or moving of cart out of specified range dimensions. Therefore, some constraints are applied in practical experimentation such as: (1) some threshold defined for magnitude of pole angle from pivot with respect to vertical dimensions and (2) distance traveled by motor cart from center [26].

Cart-pole balancing system is examined on various aspects during experimentation. Assume the pendulum is attached with a thin rod and cart moved on frictionless

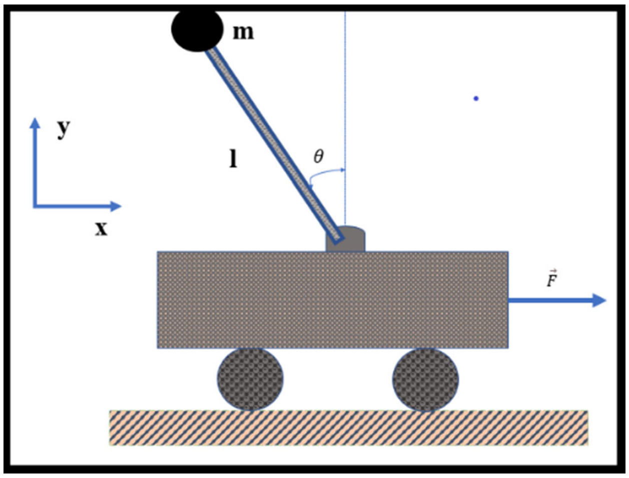

Fig. 1 Classical cart-pole problem and its associated control parameters (Adapted from [22])

surface. This system consists of two entities cart and the rod with mass m and length of rod l. The angle of deviation of pole from the vertical axis is denoted by θ in no friction condition between the cart and the surface at which cart moved due to force F. The focus is to derive the mathematical formulation of motion of cart-pole system. For this, Newton's second law applied for linear and angular movement is:

$$(M+m)\ddot{x}+\epsilon\dot{x}+ml\ddot{\theta}\cos\theta-ml\dot{\theta}^2\sin\theta=F(t) \tag{1}$$

$$ml\ddot{x}\cos\theta+\frac{4}{3}ml^2\ddot{\theta}-mgl\sin\theta=0 \tag{2}$$

In Eqs. 1 and 2, $x(t)$ is displacement in between pivot and center of mass of cart at time t, $\dot{x}$ represents linear velocity of cart and $\ddot{x}$ is cart acceleration, θ is deviation of angle of pole, $\dot{\theta}$ is angular velocity of pole and $\ddot{\theta}$ is angular acceleration, M and m being mass of cart and pendulum, respectively, $F(t)$ is force applied on cart vertically, and ϵ is viscous friction which varies within specified range. The formula for force is in Eq. (3).

$$F(t)=\alpha V(t)-\beta\dot{x}(t), \tag{3}$$

where $\alpha=\frac{K_m K_g}{Rd}$ and $\beta=\alpha^2 R$.

In Eq. 3, the notation $V(t)$ depict the voltage of cart's motors which varied to control the cart-pole system motion according to defined constraints. K_m is motor torque and back emf constant, K_g depicts gearbox ratio, R is motor armature resistance, and d shows motor pinion diameter [27].

In a similar manner, the angular acceleration of pole $(\ddot{\theta})$ and linear acceleration of cart $(\ddot{x})$ formulated in Eqs. 4 and 5, respectively, are

$$\ddot{\theta}=\frac{(M+m)gsin\theta-cos\theta[F+ml\dot{\theta}^2 sin\theta]}{\left(\frac{4}{3}\right)(M+m)l-mlcos^2\theta} \tag{4}$$

$$\ddot{x}=\frac{\left\{F+ml\left[\dot{\theta}^2 sin\theta-\ddot{\theta}cos\theta\right]\right\}}{(M+m)} \tag{5}$$

In the equation for controlling the cart-pole system, cart notations are represented as mass (M), pole mass (m), angle (θ), length (l), and force (F) input source of system [5].

3 Foundation of reinforcement learning, deep reinforcement learning

3.1 Reinforcement learning

Supervised learning, unsupervised learning and reinforcement learning are three distinguish categories of machine learning. Reinforcement learning is decision-making approach on the basis of action selected by agent and reward generated by environment corresponding to selected actions [9]. Reinforcement learning is defined as "finding suitable actions to take in a given situation in order to maximize a reward" [8]. In reinforcement learning, an agent is situated in an environment and according to taken action, environment generates reward and the next state for further processing [28]. The main factors of RL solution approach are: environment, reward and the agent as shown in Fig. 2. In RL concepts, process stared in loop from agent, agent chose some action from predefined action space and applied that action on environment. Environment generates a reward in response of action performed, and simultaneously, new state is also revealed. The generated reward defines the effectiveness of action performed by agent. In next iteration, agent received new state and reward as input from environment and objective of agent will gain optimized reward in further process, so target of achieving maximum reward in total process. This procedure will continue until final state is achieved or environment reached at its saturation point. The reinforcement learning process is visually depicted in Fig. 2 for clear understanding.

Reinforcement problem described as learning problem instead of learning system or algorithm, where agent has to learn gradually through their actions performed in environment. The behavior of reinforcement learning problem updates parameters over time span for maximizing their cumulative reward. In RL, each action selected by agent arouses an associated reward and basis of that reward agent progressively amend their action selection. In RL, agent does not get any information about action, which action should be performed to get maximize cumulative reward rather it must dig into another available option to explore the collective reward obtained from explored actions. In RL, main focus is on obtaining cumulative utmost reward rather than maximize the current reward. The best action performed by agent does not produce immediate maximum reward but may direct the environment states near that future state in which maximize reward can be obtained. So, there is no common pattern in future states and new action

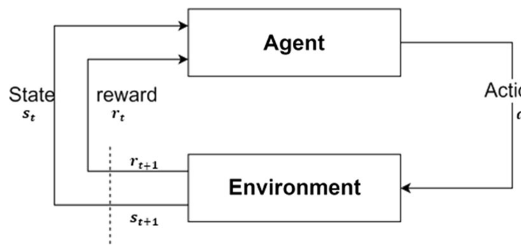

Fig. 2 A generic reinforcement learning process (Adapted from [29])

in generalize manner; the reward value is independent from each other.

RL environment follows the concepts of Markov Decision Process (MDP). MDP is represented as tuple $< S, A, P, R, \gamma >$ where S is the set of finite states, A is the set of possible actions, P is the set of transition probability, R is the set of generated reward set corresponding to action state pair, γ lies in [0,1] which is discount factor. Reward function corresponding to each state-action pair is generated by environment. In RL main goal is to maximize the reward notation with discount factor γ and cumulative reward is represented in Eq. 6 as:

$$R = \sum_{t=1}^{T} \gamma^t r_t \qquad (6)$$

In RL, the environment is defined in terms of states and reward associated with states corresponding to selected action. Existing RL research is performed on a simple environment, i.e., finite number of states, action-state pairs are limited, whereas exploration of reinforcement learning process is requisite for complex and large learning process where it is tough to attain the desired output.

3.2 Deep reinforcement learning

Deep learning computational algorithm is subdivision of machine learning compiled with neurons like structure for data storage and receiving and sending data through connection between one neuron to another, called as synapses. Neurons receive single or multiple signals, which may be raw data set or transmitted by previous layer of the neural net and after performing some computation passed to further neurons of deeper neural network. In deep neural network, one neuron is connected with more than one node with dissimilar weight which represent the influence of particular neuron's influence on overall neural network model. After receiving the input values from previous layer, accumulate each input values multiplied by associated weight and passes the value to activation function for further processing. Activation function is responsible for communication between neurons of one layer to another layer in deep learning models. Some main activation functions are threshold function, sigmoid function, Relu (Rectified Linear Unit function), hyperbola tangent function [30]. In different deep neural networks, the scalar value is the weight attached with input output mapping connection modified according to requirements of models.

In contrast to supervised and unsupervised approaches of popular deep neural network models, RL model selects the action of agent which impacts the input value on future time stamp with feedback procedure and non-static training dataset due to which multiple decision-making processes are performed for best reward computation [28]. Deep reinforcement learning, as name suggests, is the concept of deep learning and reinforcement learning merged together. This combination includes principles of reinforcement learning as concepts of Markov Property, Markov Decision process, Bellman equation, optimal policy and wide range of representation, and knowledge extracted from deep neural network features [30]. This intersection of deep neural network and reinforcement learning models especially is applicable for high-dimensional and dynamic infinite state problem statements.

3.3 Traditional reinforcement learning versus deep reinforcement learning

Deep reinforcement learning solved the shortcoming of the reinforcement learning. The main focus of deep reinforcement learning is toward maximizing the long-term reward of RL problem by learning through deep multilayered neural network instead of static value predefined in model. The deep neural network model modifies the relation between input-action pair probability, maintains action probability in nonlinear manner, update parameters to achieve the better reward value, policy progresses toward fast convergence. DRL extends the limitation of researchers by merging the features of both (deep learning and reinforcement learning) approaches by solving their shortcoming and by utilizing their properties effectively. In DRL, deep learning is all about representation through multilayer neural network dynamically without any preassumptions and RL concepts is about cumulative reward value. So Deep RL is reward-driven, and decision-making concept triggers the researcher for solving real-life unsolved problems because collectively deep learning and RL established the novel framework with emerging principles and assumption in models [28].

Consider an example of Q-learning which itself is powerful and robust method of reinforcement learning. Q-learning is considered to be scalable for large problems without introducing any excessive mathematical complexity as well. It starts with an arbitrary assumption for all Q-values and Q-table keep on updating and select policy using trial-and-error method until it reaches convergence. This action update and selection is done on random basis which may convergence solution in local optimum not in global optimum but works perfectly well for all environmental problems. Storage of Q-table for large problems and online learning problem where states change dynamically in frequent manner is a complex and challenging research issue. Exponential growth of state change and frequent change in state stages spend high consumption of memory and high amount of time for exploration. All these issues devised the concept of Deep Q learning, and associated neural network is Deep-Q-Network (DQN).

Figure 3 is discussed in three parts: (A) classical reinforcement learning, (B) classical deep learning, and (C) deep reinforcement learning. As shown in Part A of Fig. 3, in the traditional reinforcement learning, the agent interacts with environment through action corresponding to that environment generates rewards and the next state in return. The agent's main focus is toward the maximum long-term rewards achievement rather than immediate maximum reward. The tabular solution of reinforcement learning depicts the mapping between discrete states of environment and associated long-term reward with those states. In Part B Fig. 3, the supervised learning takes some unlabeled dataset either in the form of images or text data as input and should be labeled correctly with their true labels. In deep learning approaches, features as input pass through several layers in the form of artificial neurons, and with each neuron, some weight is also attached. For classifying the unlabeled input data, weights associated with each weight should be modified to achieve the ultimate goal of model. Finally, Part C of Fig. 3 is the merger of classical reinforcement learning, and deep learning is known as deep reinforcement learning in which agent opts all the features of deep neural network to solve the reinforcement learning problems. Deep reinforcement learning improves the efficiency of traditional reinforcement learning by improving representational issue.

4 Methodology: proposed deep reinforcement learning solution for cart-pole balancing

In the proposed solution for cart-pole problem, the objective is to keep pole in stable state during movement of cart. The cart can move to left and right direction due to dynamic cart's velocity in which various factors affect the stability of cart-pole dynamics. In traditional reinforcement learning model, to control the movement of agent (left/right in case of cart-pole balancing), the parameters of state s_t are taken as input at each iteration at time stamp t and corresponding environment generated reward r_t for specific action a_t performed by agent at state s_t is recorded. After performing action a_t at state s_t environment gives new state s_{t+1} along with reward, and this process continues until termination state achieved or time constraint satisfied as shown in Fig. 2 (In Sect. 3.1) which presents the working of traditional RL system.

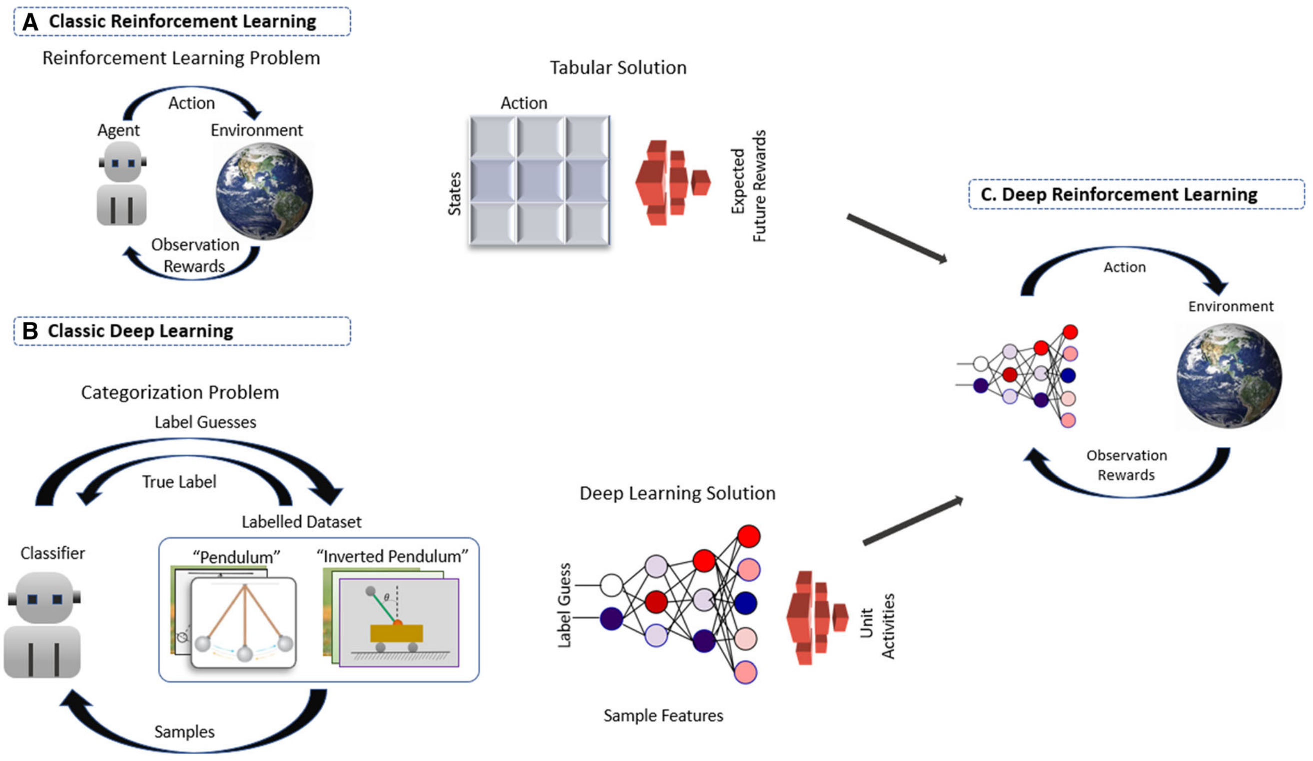

Fig. 3 Visual depiction of traditional reinforcement learning vs. deep reinforcement learning

4.1 Cart-pole problem definitions corresponding to reinforcement learning framework

The fundamental concepts of RL are generated by Markov Decision Process (See Sect. 3.1). The standard tuple set of cart-pole problem is $< S, A, P, R, \gamma >$, *where.*

- State S: It contains various state space parameters, for cart-pole problem. These parameters are cart position, cart velocity, pole angle, pole angular velocity (details in Sect. 4.2).
- Action A: Cart either pushes toward left or toward right direction.
- Transition Probability P: The probability of obtaining new state $s^{'}$ after preforming action, A at the current state S.
- Reward R: Reward R is basically a signal or some random numeric value send back by environment to agent to inform about quality of action selected and performed by agent. The value of reward may be variable or either in from of some continues functions also.
- Discount Factor γ: Through the concept of discounting, the agent's focus diverts toward not just on reward collection but also in quickest possible time. The discount factor is value between 0 and 1. If discount factor is 0, the cumulative reward is equal to immediate reward, so agent would be myopic and avaricious with respect to next time step reward only. If discount factor is near to 1, the cumulative reward gave equal importance to all the future rewards, so agent is more far sighted.

The detailed explanation of all the factors of MDP framework defined for cart-pole balancing problem is discussed in further subsections.

4.2 State space parameter (SSP)

A. The cart pole is defined by 4-dimensional state space $S = (\theta, x, \dot{\theta}, \dot{x})$, where θ is the angle of the pole, x is the position of the cart, $\dot{\theta}$ is the pole velocity, and $\dot{x}$ is the cart velocity (discussed in detail in Sect. 2). Out of all SSP, two factors angle, angular speed are factors for pole and remaining two, horizontal velocity and cart position are factors for cart.

B. Angle (θ): It is the degree between stable/center position of pole to the actual position of pole while movement.

C. Angular velocity ($\dot{\theta}$): It shows how fast pole rotates around its pivot point under predefined constraints.

D. Position (x): Horizontal position of the cart on the track

E. Horizontal Velocity ($\dot{x}$): It depicts the cart's x-axis velocity horizontally

The considered ranges for state space parameters for defined 4-dimensional cart-pole problem are listed in Table 1. The cart position (x) ranges in between -4.8 to $+4.8$, pole angle lies in $\pm 24°$, cart velocity and pole angular velocity are $-\propto$ to $+\propto$.

4.3 Action

Reinforcement learning is based on trial-and-error response to actions and rewards associated with selected action issued by environment, and this makes cart-pole system suitable model for RL approach due to its dynamic nature. In the cart-pole system agent is either a controller or an algorithm responsible for cart movement in left and right direction. The action was performed on cart in the form of various forces or torques, and based on that, cart moved in left or right direction horizontally. The environment is predefined physically constrained area of the system, and the reward is factor which motivates the cart-pole system to achieve affirmed in its balanced position.

The cart-pole agent is confined to two possible actions: apply a constant force on cart in rightward direction and another is exerting a left ward force on cart as shown in Fig. 4.

4.4 Reward

In the presented work, cart-pole problem is experimented using deep reinforcement learning approach for two reward functions—mean squared loss (MSE) function and Huber loss function.

4.4.1 Mean square error loss

Mean-squared error (MSE) as name suggested is a loss function which returns value of a mean of the square of the errors. So, to compute the MSE the difference between model's prediction and the ground value is calculated and then square it and then average it.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{7}$$

where y_i and $\widehat{y}_i$ are the predicted value of model and true value, respectively. N is the number of samples.

Table 1 SSP for experimented cart-pole problem

State space parameter	Values
Cart position (x)	[− 4.8, + 4.8]
Cart velocity ($\dot{x}$)	$[-\infty, +\infty]$
Pole angle (θ)	[− 0.418 rad (− 24°), + 0.418 rad (24°)]
Pole angular velocity $\left(\dot{\theta}\right)$	$[-\infty, +\infty]$

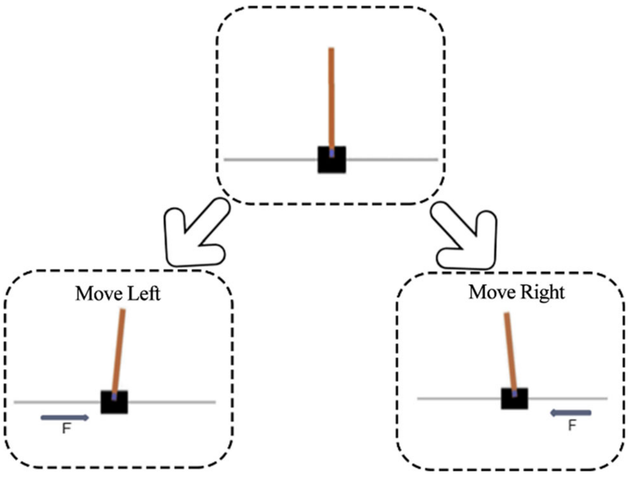

Fig. 4 Action performed by cart pole based on state parameter setting

4.4.2 Huber loss

Mean-squared error loss function depicts outstanding learning for outlier in dataset, while man absolute error (MAE) solves this issue by avoiding the outliers. Henceforth, Huber loss function offers the balance between MSE and MAE loss function. Huber loss has concepts of both MSE and MAE as when error is small MSE applied otherwise MAE applied.

$$\mathcal{L}_{\delta}(y,f(x) = \begin{cases} \frac{1}{2}(y-f(x))^2 \; for \; |y-f(x)| \le \delta, \\ \delta|y-f(x)| - \frac{1}{2}\delta^2 \; otherwise. \end{cases} \tag{8}$$

Here, δ is the hyperparameter defining the range for MSE and MAE. When the error value is less than delta, the error is quadratic; else, it is absolute error. These concepts combined the best of both loss function efficiently. The algorithm for Huber Loss is shown in Algorithm 6.

4.5 Deep Q learning

4.5.1 Q learning

Q learning is off-policy control optimal method where the samples (S, A, R, S') are generated by an exploratory policy in order to maximize Q (S', A') for obtaining a deterministic optimal target policy. The Q Learning algorithm and its step-by-step process for achieving maximum Q (S', A') are written in Algorithm 1. In the SARSA (standard Q-Learning Process) [29] defined as values S, A, R, S', A' generated by ε-greedy policy, instead in Q-Learning exploration with ε-greedy policy is used for generating the samples (S, A, R, S').

$$Q(S,A) = R(S,A) + \gamma max_A Q(S',A) \tag{9}$$

In Eq. (9), the cumulative reward which is also known as Q-value yield as immediate reward R (S, A) by performing action A at state S plus the maximum Q-value from the next state S' and gamma is discount factor responsible for contribution of future rewards which either diminish the contribution of future reward or increase it. The learning algorithm for the same process is defined in Algorithm 3, and the corresponding action selection process is defined in Algorithm 2. Q (S', A) depends on future state values as Q (S'',A) with some coefficient of gamma as shown in Eq. (10):

$$Q(S,A) \rightarrow \gamma Q(S',A) + \gamma^2 Q(S'',A)..........\gamma^n Q(S''^{....n},A) \tag{10}$$

In Q-learning, the target is to find the Q maximizing action $argmax_{A'}$ Q(S', A') in state S' with exploitation, and this maximum reward achieving process is in Algorithm 4. To update the Q-values rule is defined in Eq. (11):

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha * [R_{t+1} + \gamma * max_{A_{t+1}} Q(S_{t+1},A_{t+1}) - Q(S_t,A_t)] \tag{11}$$

Q learning is distinguished with SARSA in terms of policy as Q-learning is based on off-policy leaning which involves two different policies behavior policy $b\ (a|\ s)$ for exploration purpose and to generate examples and another target policy $\pi(a|s)$ through which agent learned as optimal policy. Samples are generated by the behavior policy used for training the agent that increases the learning efficiency of approach, and this procedure is known as *experience replay*. In experience replay, the agent interacts with environment through some action and achieve experiences in the form of reward and replay these obtained experiences multiple times for learning purpose of agent. These experiences generated by exploration behavior policy stored in format of samples (s, a, r, s', done) in buffer and further used in deterministic target policy using Q-values. Process for learning parameter and updating parameters are presented in Algorithm 3 and Algorithm 5, respectively. For this continues learning previous samples from behavior policy used and stored in *replay buffer*. The fixed size buffer keeps removing the older samples stored for storing the latest samples in replay buffer which makes the learning process more efficient due to several time utilizations of samples for learning purpose [29].

```
Algorithm 1: Q- Learning Algorithm
1.  Input: $\alpha$ learning rate, $\epsilon$ : a small number, $\gamma$ : discount factor
2.  Initialization
    2.1.  Initialize Q (s, a) arbitrarily,
    2.2.  Q(terminal)←0
3.  For each sample do
    3.1.  Initialize state S
    3.2.  While round < max_round
    3.3.  Do
          3.3.1. A ← CHOOSE_ACTION (Q, S, $\epsilon$)
          3.3.2. Select action within available action in the current state according to highest Q-value in
                 the next-state
          3.3.3.LEARN (S, A, R, S')
          3.3.4. S ← S'
    3.4.  While S is not terminal
```

```
Algorithm 2: CHOOSE_ACTION (Q, S, $\epsilon$)
1.  Input: Q: Q table generated so far, $\epsilon$ : a small number, S: current state
2.  Result: Select action A
3.  Function CHOOSE_ACTION (Q, S, $\epsilon$)
    3.1.  N ← uniform random number between 0 and 1
    3.2.  If N > $\epsilon$:
          3.2.1. maxQ ← GET_MAXQ(S)
          3.2.2. A ← choose action from action space based on maxQ
    3.3.  Else:
          3.3.1. A←random action from the action space
    3.4.  UPDATE_PARAMETER ()
    3.5.  Return selected action A
```

```
Algorithm 3: LEARN (S, A, R, S')

1.  $Q(S,A) \leftarrow Q(S,A) + \alpha * (\text{HUBER\_REWARD}() + (\gamma * \text{GET\_MAXQ}(S')) - Q(S,A))$
```

```
Algorithm 4: GET_MAXQ(S):
1.  Return the maximum Q- value of all actions based on the current state S'
```

```
Algorithm 5: UPDATE_PARAMETER ()
1. Update $\epsilon$ and $\alpha$ after each action else set them 0 if not learning
```

```
Algorithm 6: HUBER_REWARD ():
1. If it is small error
    1.1. Then R=error$^2$ /2
2. Else
    2.1. R= delta*(|error|-delta/2)
3. Return R
```

4.5.2 Deep Q network (DQN)

Q learning suffered with problem of dimensionality means it requires discrete state space if state space is continuous in nature two main issues depicts: (1) memory required for saving/updating of large dimension state-action samples in Qtable and (2) computation time for exploration of Q table. Deep Q learning solves this problem by using neural network to estimate Q-value function. In the Deep Qnetwork, state is given as input and resultant Q-value is generated of all possible action performed by agent. The Deep Qlearning approach applied for cart-pole balancing is detailed in Algorithm 7 which consist of all the input parameters and learning process incorporated with reinforcement Q-learning algorithm for superior outcomes. A sequential Feed-Forward Neural Network model has been used for learning process to train generated state-action pair. The GET_-MODEL() algorithm (Algorithm 8) outlines learning process. The elaborated DQN process is as follows:

(1) In DQN, state s takes as input and in return the Q-value of all possible actions in the given input state.

(2) Chose the action on input state s by using epsilon-greedy policy which based on concepts as select a random action with maximum Q- value.

(3) Move to next state S', after performing selected action on state s and this sample is stored in replay buffer as S, A, S', A'

(4) Repeat the procedure for some random batches from the replay buffer and compute the reward.

(5) In DQN, the loss function is as follows:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}\left[r_i + \left((1 - done_i).\gamma.max_{a_i'}\hat{q}\left(s_i', a_i'; w_t^-\right)\right) - \hat{q}(s_i, a_i; w_t)\right]^2 \tag{12}$$

Further, the gradient of L with regard to w is computed, and the computed gradient value updates the weights of w of the online network. The equation for this is as follows:

$$\nabla_w \mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}[r_i + ((1 - done_i).\gamma.\max_{a'_i}\hat{q}(s'_i, a'_i ; w_t^-)) - \hat{q}(s_i , a_i ; w_t)]\nabla\hat{q}(s_i , a_i ; w_t)^2 \tag{13}$$

$$w_{t+1} \leftarrow w_t - \alpha\nabla_w\mathcal{L} \tag{14}$$

(6) After each iterations, update the weight of network and repeat steps for N no of episodes [31].

Algorithm7: Deep Q-learning Algorithm

```
1. Input: $\alpha$ learning rate, $\epsilon$ : a small number, $\gamma$ : discount factor
2. Initialization:
   2.1. Initialize action- value function Q (S, A) with random value
   2.2. Initialize target action-value function $\hat{Q}\ (S, A)$ with random value
   2.3. Initialize q_network and target_network using GET_MODEL ()
3. For each Sample do:
   3.1. Initialize state S
   3.2. While round < max_round
   3.3. Do
        3.3.1. Select a random action A based on $\epsilon$
        3.3.2. Otherwise select A ← CHOOSE_ACTION (Q, S, $\epsilon$)
        3.3.3.LEARN (S, A, R, S')
        3.3.4.S← S'
   3.4. While S is not terminal
   3.5. After each round copy value from Q to $\hat{Q}$
   3.6. Repeat step 3 for each Sample
```

Algorithm 8: GET_MODEL ()

```
1. Build the Deep Q Network with input layer of dimension 4 with hidden layer and activation function
   and output layer of dimension 2
```

Table 2 Hyperparameters for best variant of Q learning and DQN for cart-pole problem

Parameter	Value
Gamma	0.5
Learning rate	0.5
Epsilon	1
Rounds	10
Samples	400
Input neuron	4
Output neuron	2
Hidden layers	2
Hidden neuron	24 (hidden layer 1), 12(hidden layer 2)
Activation functions	Relu, linear

4.5.3 Double deep Q network

The Deep Q learning algorithm is positive bias which distort the combined reward due to overestimation phenomena of action value. So, to solve this overestimation problem the dual network structure is introduced with small alteration in DQN, known as Double DQN (DDQN). This same Double DQN process in form of algorithm is detailed in Algorithm 9 where feed forward sequential neural network as learning model is used two time in step 2.2 and step 2.3 of algorithm. This dual network structure reduces the overestimation phenomenon effectively in the training process. The DDQN architecture includes a target network, experience replay and the deep Q network. In DDQN, Q-value computed as Eqn. (15)

$$Q(s_t\ ,\ a_t\ )\ =\ r_j\ +\ \gamma\ \max_a \hat{Q}\ (\ S_{t+1}\ ,\ argmax_a\ Q(s_{t+1}\ ,\ a\ ;\ \theta)) \tag{15}$$

Deep Q-Network select the best possible action a to compute the maximum Q value of next state s_{t+1} , then the target network utilizes the maximum action for further calculation of Q-value.

```
Algorithm 9: Double DQN algorithm
1.  Input: $\alpha$ learning rate, $\epsilon$ : a small number, $\gamma$ : discount factor
2.  Initialization:
    2.1. Initialize an empty replay memory D
    2.2. Initialize online network Q using GET_MODEL () with random value
    2.3. Initialize target network using $\hat{Q}$ GET_MODEL () with random value
3.  For each Sample do:
    3.1. For each episode do:
        3.1.1. Reset CartPole_environmnet
        3.1.2. For each step do:
            3.1.2.1. A ← CHOOSE_ACTION (Q, S, $\epsilon$)
            3.1.2.2. Select action within available action in the current state according to highest Q-value in the
                     next-state
            3.1.2.3. Add experience [state, action, reward, next_state] in replay memory D
            3.1.2.4. If len (replay_memory)> Min_Memory_For_Experince_Replay:
                3.1.2.4.1. Sample a random batch of experinces[state, online_net_selected_action, target_q_value,
                           next_state] from D.
            3.1.2.5. If next_state is terminal do:
                3.1.2.5.1. Q_update= reward
            3.1.2.6. Else
                3.1.2.6.1. Q_update= reward +
                           $Q(s_t, a_t) = reward + \gamma \max_a \hat{Q}\ (S_{t+1}, argmax_a\ Q(s_{t+1}, a;\theta)$
        3.1.3. Repeat until S is not terminal
    3.2. After each episode copy value from Q to $\hat{Q}$
    3.3. Repeat step 3 for each Sample
```

5 Experimental setup and experimental outcome

The results are presented in the form of reward function achieved using Q learning and Deep Q learning for both loss functions: MSE loss and Huber Loss. The training process was lengthy, and performance has been validated with varying hyperparameter setting, and tweaking of parameter has been done in order to speed up the convergence. The hyperparameters for the best variant of Q learning and DQN are stated in Table 2. Gamma, learning rate, Epsilon, rounds, and samples are common parameter in Q learning and DQN. For every transformation of state-action pair in cart-pole balancing problem, calculate distance from target stable position which is measured as loss function. Further stochastic gradient descent is used to update the position by minimizing the loss in respect to model parameter. Feed-forward training model has been used in DQN for convergence of cart-pole balancing problem. The hyperparameter setting for the same is listed in Table 2.

5.1 Performance evaluation of cart-pole balancing problem

The performance of cart-pole balancing is evaluated on two factors: one is reward function and the other is Deep Q learning algorithms performance comparison. The MSE and Huber loss function are the reward functions that are used to measure performance of Q learning, DQN, and Double DQN. This shows that the minimum loss will be considered as best reward while measuring performance of applied algorithms. As per our knowledge, Huber loss has not been applied yet on cart-pole balancing problem.

Firstly, MSE results (Sect. 5.1.1) for cart-pole problem are discussed where Q learning, DQN, and Double DQN results for MSE loss are presented in Figs. 5, 6, and 7, respectively. Secondly, Huber loss results (Sect. 5.1.2) are discussed along with average reward function graph plot. Figures 8, 9, and 10 show the Huber Loss outcome for Q learning, DQN, and Double DQN, respectively, for cart-pole balancing problem. Finally, performance comparison of considered reward functions (MSE and Huber) for both deep Q Learning Algorithms is discussed in Sect. 5.1.3 which reflects a clear picture of efficacy of cart-pole balancing research problem using Huber Loss function as compared to MSE Loss function.

5.1.1 MSE reward result comparison for Q learning, deep Q network, and double deep Q network

The outcome presentation of MSE of Q learning interaction is shown in Fig. 5. In the beginning, the training of built cart-pole environment is effective and agent is exploring the environment which can be seen in transition and action

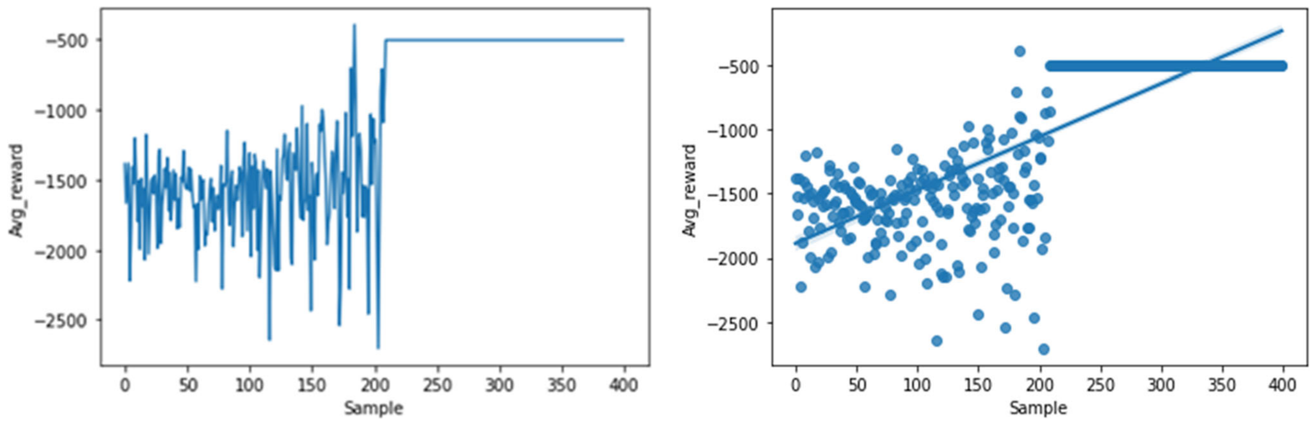

Fig. 5 Mean square error loss Q learning results (sample vs. average reward)

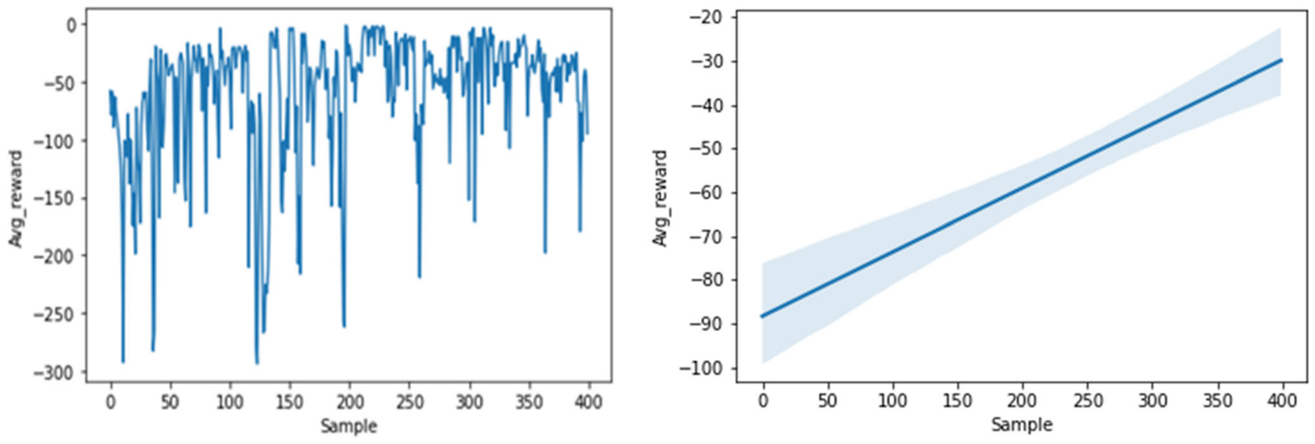

Fig. 6 Mean square error loss DQN learning results (sample vs. average reward)

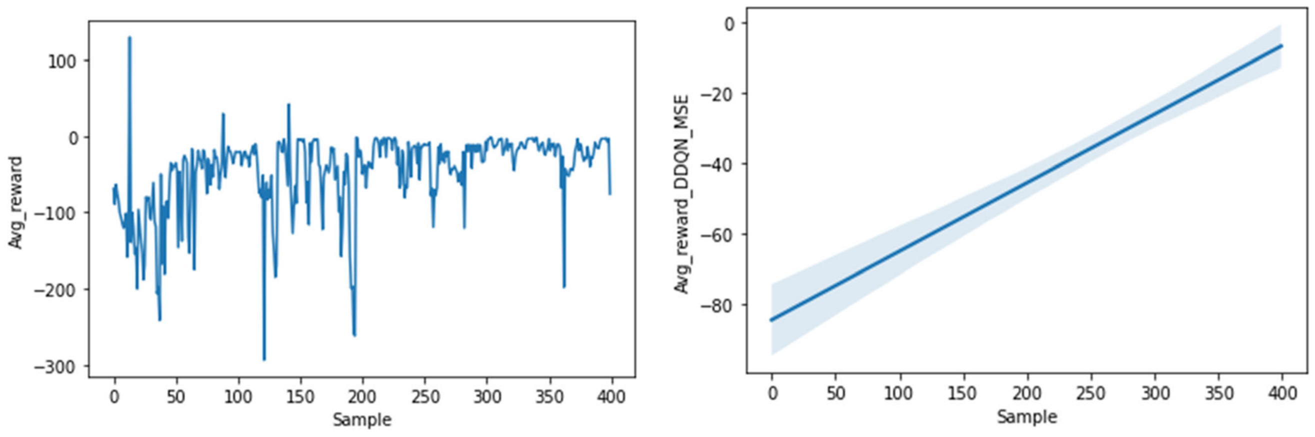

Fig. 7 Mean square error loss double deep Q network learning results (sample vs. average reward)

behavior in reward graph (see Fig. 5). But approximately after 200 iterations, our agent is stuck without even trying to behave differently. This is basically known as exploration versus exploitation dilemma. The MSE loss value of Q-Learning for cart pole is stuck to − 500 average reward which is almost in constant state from 200 to 400 iterations.

This bad exploration of Q learning is improvised in DQN as depicted in DQN Average reward plot in Fig. 6. The MSE loss with DQN is reaching up to the average reward value of – 30 in 400 iterations. Environment is continuous in exploration stage, and experience is helping in achieving better reward. The initial performance of Q

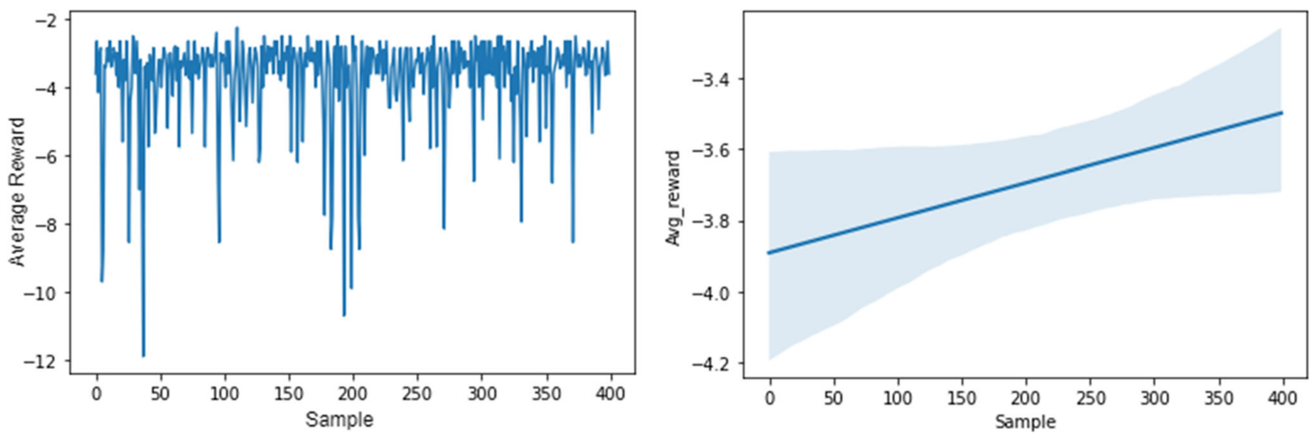

Fig. 8 Huber loss Q learning results (sample vs. average reward)

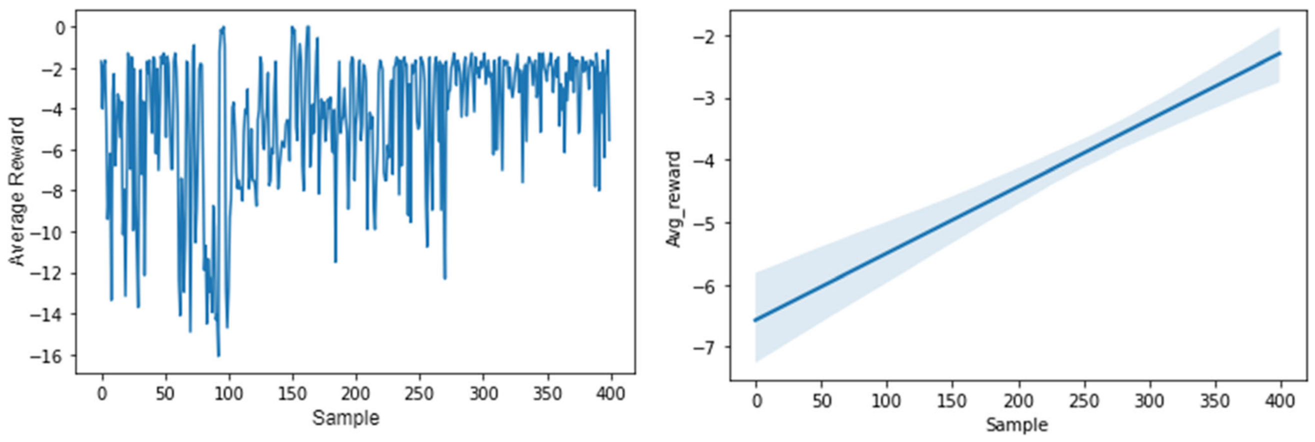

Fig. 9 Huber loss DQN learning results (sample vs. average reward)

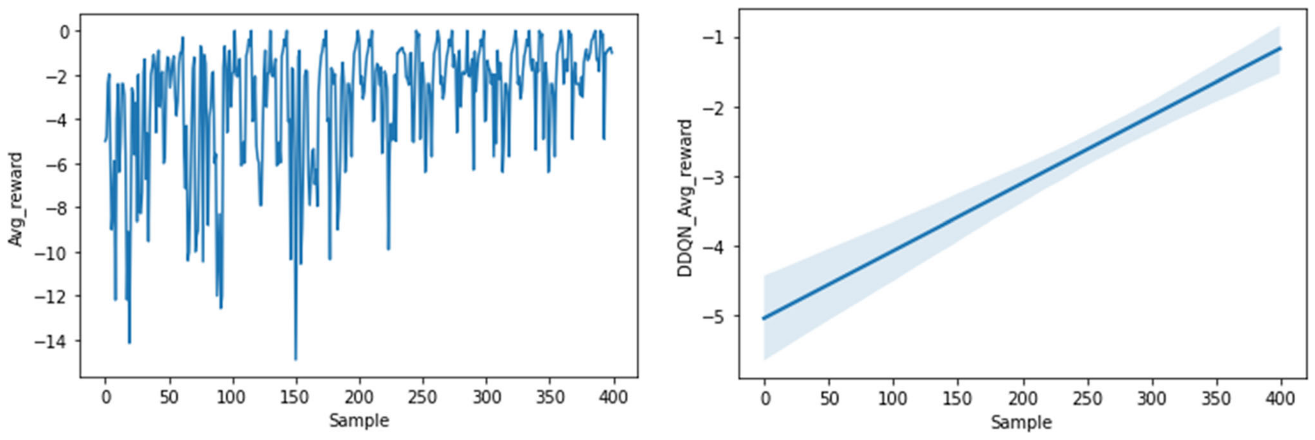

Fig. 10 Huber loss double DQN learning results (sample vs. average reward)

learning is good, but the convergence of DQN to achieve higher reward is commendable as compared to Q learning due to fast convergence. Even if we compare outcome of 200 iterations, then it is seen in the reward graph that during the training, average attained reward in DQN is – 63, whereas for the same iterations, − 500 is the attained average reward by Q learning. Hence, in the same number of episodes, DQN is able to achieve much better results as compared to Q learning for MSE loss.

The MSE loss of Double DQN is depicted in Fig. 7 where it is clearly reflecting that loss is reducing in Double DQN reinforcement learning approach. Although the

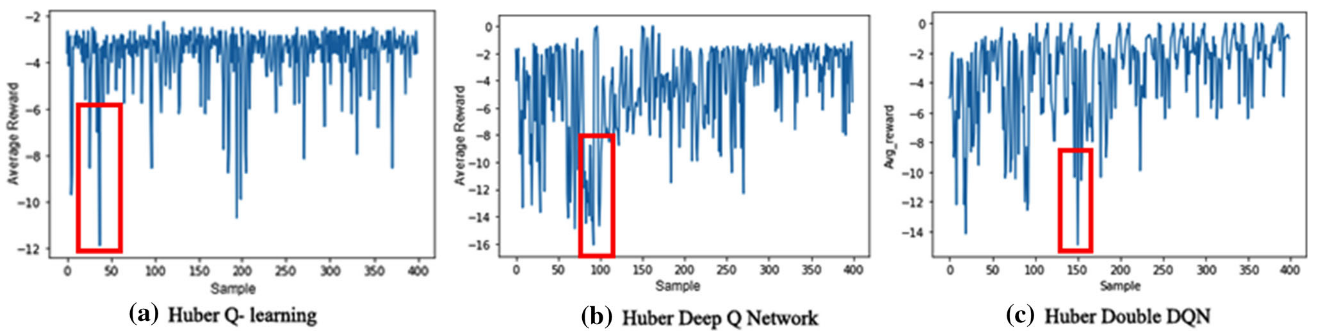

Fig. 11 Effect of uncontrolled disturbance on Reward **a** Huber Q Learning, **b** Huber Deep Q Network, and **c** Huber Double Deep Q Network

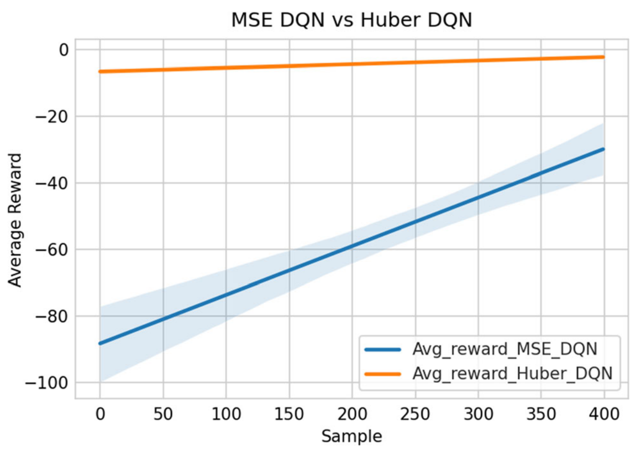

Fig. 12 Comparison reward plot of Huber loss and MSE loss for DQN

difference in result improvement is not striking, results are better. Results of the same 400 iterations are shown which shows narrow exploration toward final iterations. Convergence toward improved rewards is quite observant in Double DQN result for cart-pole balancing. The average attained reward for cart-pole balancing problem using Double DQN is − 48 which ranges from − 85 to − 15 for MSE loss.

5.1.2 Huber loss reward result comparison for Q learning and deep Q network

The results of Huber for Q learning interaction are shown in Fig. 8. The training of cart-pole environment using Huber Q learning shows the performance improvement in the range of − 12 (minimum reward) to − 3.6 (maximum reward) in 400 samples. Here performance of Q learning using Huber loss is considerable, but convergence is slow. Right side plot of Fig. 8 shows that Huber loss average reward plot is constantly improving performance of training using Q learning and ranges from – 3.9 to – 3.6 in 400 samples which is staggeringly poor convergence.

The Huber loss in DQN is performing best training and achieving minimum loss of – 2. Even as can be seen in left side plot of Fig. 9, at sample 100 Huber loss is 0 which means pole at balance and stable position, after sample 100 results are far improved. In case of comparison of Huber

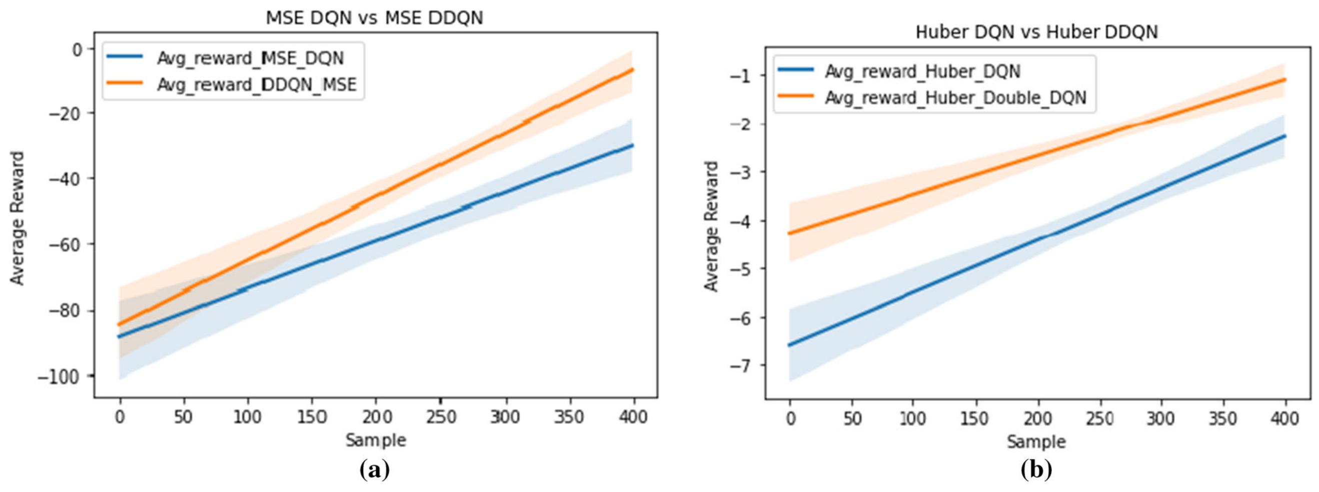

Fig. 13 Comparison of reward plot for double DQN: **a** MSE loss reward function **b** Huber loss reward function

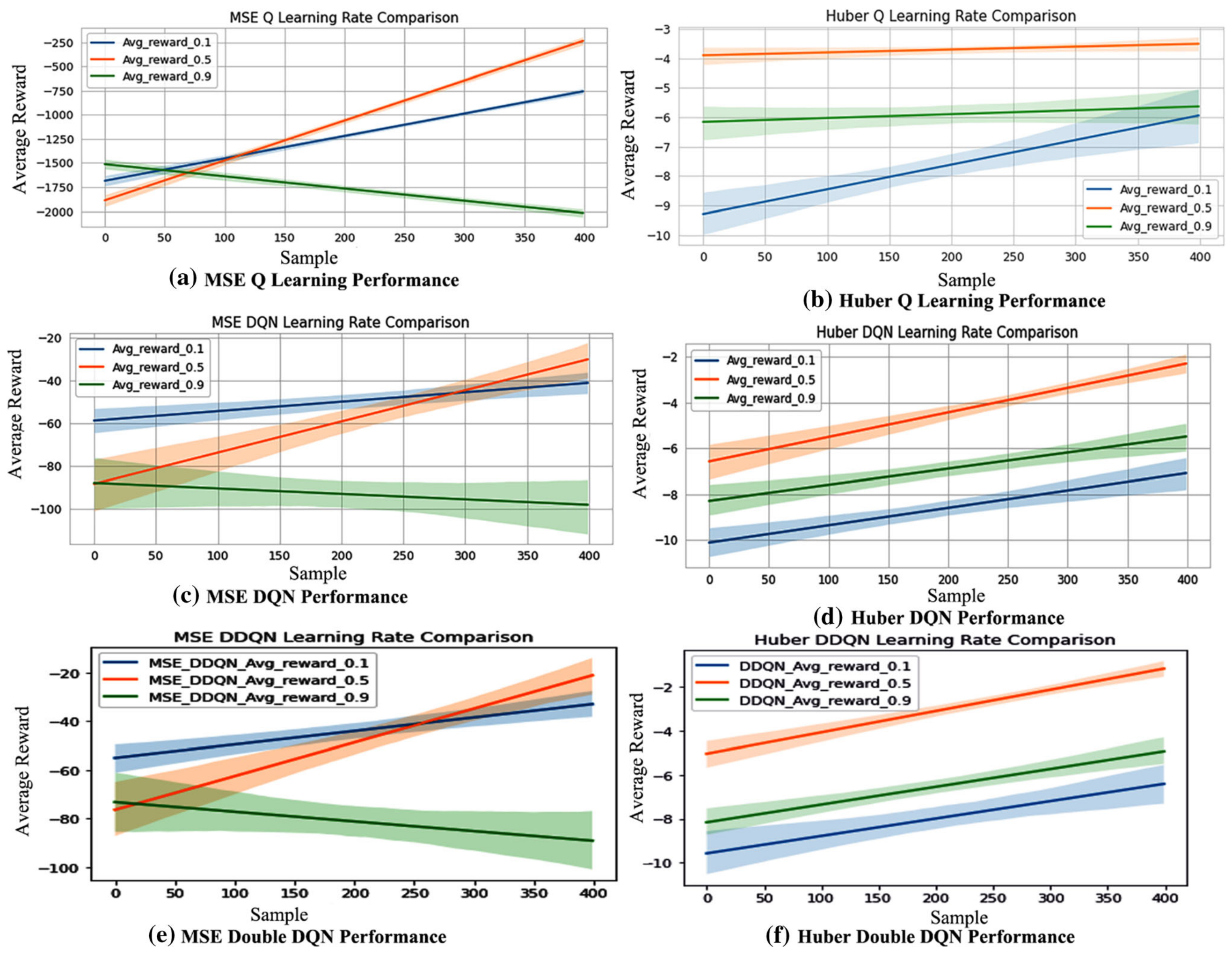

Fig. 14 Comparison plots of hyperparameter learning rate for learning Rate 0.1 (green color), 0.5 (orange color), 0.9 (blue color) (Color figure online)

loss in Q learning and DQN, performance of DQN is toward fast convergence. Initial outcome of Q learning was better than DQN and training starts with Huber loss – 3.9 which was − 7 in case of DQN, but DQN is continuously exploring the varying environment factor which can be seen by transition behavior in reward plot. That is the reason of achieving − 2 reward in 400 iterations.

The Huber Loss outcome for Double DQN loss is shown in Fig. 10 where cart-pole balancing loss is reduced to − 1.5 which was − 2 while applying DQN with Huber loss. Not much difference is achieved through Double DQN with Huber same like seen in case of Double DQN results with MSE loss. Results are marginally improved which shows notable performance of cart-pole balancing problem using Double DQN with Huber Loss.

The uncontrolled disturbance (friction, wind, etc.) due to environmental change can impact system's functionality and incorporate insensitivity to the cart-pole system which majorly affect robustness of any supervised and unsupervised model. That's why Deep Reinforcement Learning (DRL) fetched researchers' attention despite the complexity in system design. DRL is considered as the most robust and resilient solution for real-life and experience-based learning systems [35]. The applied DRL approaches-Q-Learning, Deep Q Network, Double Deep Q Network strengthen the system by handling the sudden environmental damages while training the model and maintain robustness. Basically, uncontrolled disturbance such as friction (in rainy weather) will be a scenario for experienced learning which changes in cart-pole balancing condition but system will not be suffered due to this partial damage. Sudden descend in reward is prominent in visual depiction of Q Learning, Deep Q Network and Double Deep Q Network (shown in Fig. 1); these changes could be due to partial damage. This partial damage changes reward to undesired level as shown by red color boxes in Fig. 11, but learning rate is not diverted due to fall in reward. On the other hand, in case of supervised or unsupervised machine learning, system will consider these factors and affect robustness and resilience of cart-pole balancing system drastically. A dynamic and optimized algorithm—reinforcement learning, heuristic and meta-heuristic learning algorithms, will be the best for real-time and experienced learning-based research problems.

5.1.3 Comparison in Huber Loss and MSE Loss Reward Results of DQN and Double DQN

A comparison to showcase best outcome achieved is discussed in this section. It is clear from Sects. 5.1.1 and 5.1.2 that DQN is far better than Q learning and marginal improvement in cart-pole balancing is achieved through Double DQN as compared to DQN. Hence, comparison of DQN outcome for MSE Loss and Huber loss is presented in Fig. 12. In case of DQN, performance of MSE is subsided in comparison with Huber loss. See Fig. 12, blue line presents the reward plot of MSE and orange line presents the reward plot of Huber for DQN which shows that Huber loss is almost negligible as compared to MSE loss for cart-pole balancing problem.

Another comparison of average rewards of MSE DQN and MSE Double DQN is shown in Fig. 13a. The outcome reflects that results are improved, and loss has been reduced by using Double DQN. Despite the fact that performance does not have drastic and remarkable improvement. Similar kind of results is seen while comparing DQN and Double DQN performance outcome using Huber Loss (See Fig. 13b).

5.1.4 Hyperparameter tuning

The results of learning rate hyperparameter settings are shown for tuning parameter understanding purpose. A comparative reward plot for 0.1, 0.5, and 0.9 learning rate for Q Learning with MSE and Huber Loss reward plots are shown in Fig. 14a and b, respectively. Further, comparative reward plot for the same learning rate for DQN with MSE and Huber loss reward plots are shown in Figs. 14c and d, respectively. The comparative reward plot for Double DQN for the same learning rate is shown in Fig. 14e and f for MSE and Huber Loss, respectively. These learning rate hyperparameter results depict the superior performance of cart-pole balancing problem with learning rate 0.5 as compared to 0.1 and 0.9 learning rate in all the considered cases. Similarly, varying parameter setting in Q learning and DQN has been implemented, and finally, best variant outcome is shown in this research paper. The hyperparameters for best variants are listed in Table 2.

6 Concluding remarks and future scope

In this paper, reinforcement learning and deep reinforcement learning have been applied to solve real-world cart-pole balancing problem. Specifically, we first presented the theoretical foundation of cart-pole balancing problem, its application areas such as robotics, reinforcement learning, and deep reinforcement learning. Further, cart-pole balancing issue with reference to reinforcement learning parameters has been introduced for researchers' viewpoint which help future research work to understand the concept and parameter setting of cart-pole balancing research issue according to reinforcement learning concepts. Finally, methodology has been discussed where RL and DRL are validated. Some recent research works have already applied mean square error as reward function for cart-pole balancing problem. In this paper, a novice reward function Huber loss is implemented and tested to set up cart-pole balancing simulation. As per our knowledge, Huber loss has never been applied for cart-pole balancing problem, whereas the outcome shows tremendous performance of deep Q Learning by using Huber loss. A comparative outcome of Q learning with Deep Q learning for both MSE and Huber loss is defined in results. Results showed the extraordinary effectiveness of Huber loss reward function in DQN.

In the future, study will be toward.

- Validating this same proposed methodology on other real-life case studies.
- Study on reward functions to get better outcome suggested for cart pole and other physical object properties detection problems.
- Exploration of other learning algorithm in deep Q network in comparison with applied feed-forward sequential neural network.

Declarations

Conflict of interest The authors have no conflicts of interest to declare that are relevant to the content of this article.

References

1. Chen Y, Han X (2021) Four-rotor ae flight of inverted pendulum based on reinforcement learning. In: 2021 2nd international conference on artificial intelligence and information systems, pp 1–5
2. Moreira I, Rivas J, Cruz F, Dazeley R, Ayala A, Fernandes B (2020) Deep reinforcement learning with interactive feedback in a human–robot environment. Appl Sci 10(16):5574

3. Nguyen HS, Cruz F, Dazeley R (2021) A broad-persistent advising approach for deep interactive reinforcement learning in robotic environments. ArXiv preprint arXiv:2110.08003
4. Variengien A, Nichele S, Glover T, Pontes-Filho S (2021) Towards selforganized control: using neural cellular automata to robustly control a cart-pole agent. arXiv preprint arXiv:2106.15240
5. Nagendra S, Podila N, Ugarakhod R, George K (2017) Comparison of reinforcement learning algorithms applied to the cart-pole problem. In: 2017 international conference on advances in computing, communications and informatics (ICACCI), pp. 26–32, IEEE
6. Prasad LB, Tyagi B, Gupta HO (2014) Optimal control of non-linear inverted pendulum system using pid controller and lqr: performance analysis without and with disturbance input. Int J Autom Comput 11(6):661–670
7. Haydari A, Yilmaz Y (2020) Deep reinforcement learning for intelligent transportation systems: a survey. IEEE Trans Intel Transp Syst
8. Sutton RS, Barto AG (1998) Reinforcement learning: an introduction MIT press. Cambridge, MA 22447 (1998)
9. Littman M, Moore A (1996) reinforcement learning: a survey, journal of artificial intelligence research 4. syf
10. Yaghmaie FA, Ljung L (2021) A crash course on reinforcement learning. arXiv preprint arXiv:2103.04910
11. Manrique Escobar CA, Pappalardo CM, Guida D (2020) A parametric study of a deep reinforcement learning control system applied to the swing-up problem of the cart-pole. Appl Sci 10(24):9013
12. Lapan M (2018) Deep reinforcement learning hands-on: apply modern rl methods, with deep Q-networks, value iteration, policy gradients, TRPO. Packt Publishing Ltd, AlphaGo Zero and More
13. Sharma S (2020) Modeling an inverted pendulum via differential equations and reinforcement learning techniques
14. Xie A, Finn C (2021) Lifelong robotic reinforcement learning by retaining experiences. arXiv preprint arXiv:2109.09180
15. Cruz F, Dazeley R, Vamplew P, Moreira I (2021) Explainable robotic systems: understanding goal-driven actions in a reinforcement learning scenario. Neural Comput Appl, 1–18
16. Zhang Z, Liniger A, Dai D, Yu F, Van Gool L (2021) End-to-end urban driving by imitating a reinforcement learning coach. In: Proceedings of the IEEE/CVF international conference on computer vision, pp 15222–15232
17. Zhu B, Bedeer E, Nguyen HH, Barton R, Henry J (2021) Uav trajectory planning in wireless sensor networks for energy consumption minimization by deep reinforcement learning. IEEE Trans Veh Technol 70(9):9540–9554
18. Bignold A, Cruz F, Dazeley R, Vamplew P, Foale C (2022) Human engagement providing evaluative and informative advice for interactive reinforcement learning. Neural Comput Appl 1–16
19. Wei P (2020) Exploration-exploitation strategies in deep q-networks applied to route-finding problems. In: Journal of physics: conference series, vol 1684, p 012073 (2020). IOP Publishing
20. Mukherjee A (2021) A comparison of reward functions in q-learning applied to a cart position problem. arXiv preprint arXiv:2105.11617
21. Bates D (2021) A hybrid approach for reinforcement learning using virtual policy gradient for balancing an inverted pendulum. arXiv preprint arXiv:2102.08362
22. Kumar S (2020) Balancing a cartpole system with reinforcement learning—a tutorial. arXiv preprint arXiv:2006.04938
23. Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J, Zaremba W (2016) Openai gym. arXiv preprint arXiv:1606.01540
24. Bignold A, Cruz F, Dazeley R, Vamplew P, Foale C (2021) Persistent rule-based interactive reinforcement learning. Neural Comput Appl 1–18
25. Stimac AK (1999) Standup and stabilization of the inverted pendulum. PhD thesis, Massachusetts Institute of Technology, Department of Mechanical Engineering
26. Kafetzis I, Moysis L (2017) Inverted pendulum: a system with innumerable applications. School Math Sci
27. Landry M, Campbell SA, Morris K, Aguilar CO (2005) Dynamics of an inverted pendulum with delayed feedback control. SIAM J Appl Dyn Syst 4(2):333–351
28. Botvinick M, Wang JX, Dabney W, Miller KJ, Kurth-Nelson Z (2020) Deep reinforcement learning and its neuroscientific implications. Neuron 107(4):603–616
29. Gym O, Sanghi N Deep reinforcement learning with python
30. Lei C (2021) Deep learning basics. In: Deep learning and practice with mindspore, pp 17–28. Springer
31. Choudhary A (2019) A hands-on introduction to deep q-learning using openai gym in python, Dostupńe tiěz z: https/www Analytics vidhya. com/blog/2019/04/introduction-deep-qlearning-python/[online], cit.[2020–12–10]
32. Wang F, Qian Z, Yan Z, Yuan C, Zhang W (2019) A novel resilient robot: kinematic analysis and experimentation. IEEE Access 8:2885–2892
33. Xue L, Liu CJ, Lin Y, Zhang WJ (2015) On redundant human-robot interface: concept and design principle. In: 2015 IEEE international conference on advanced intelligent mechatronics (AIM) (pp 287–292), IEEE
34. Zhang W, Yang G, Lin Y, Ji C, Gupta MM (2018) On definition of deep learning, 2018 world automation congress (WAC)
35. Zhang WJ, Lin Y (2010) On the principle of design of resilient systems–application to enterprise information systems. Enterprise Inf Syst 4(2):99–110