

CART POLE SYSTEM ANALYSIS AND CONTROL USING MACHINE LEARNING ALGORITHMS

A DESSERTATION

**SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE AWARD OF
THE DEGREE OF**

**MASTER OF TECHNOLOGY
IN
CONTROL & INSTRUMENTATION**

Submitted by:

**FAIZ MOHAMMAD ALI
Roll No. 2K20/C&I/01**

**UNDER THE SUPERVISION OF
PROF. MADHUSUDAN SINGH**



ELECTRICAL ENGINEERING DEPARTMENT

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

MAY 2022

ELECTRICAL ENGINEERING DEPARTMENT
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

CANDIDATE'S DECLARATION

I, Faiz Mohammad Ali, Roll No. 2K20/C&I/01, an M.Tech (Control & Instrumentation) student, hereby declare that the MAJOR PROJECT titled "**Cart pole system analysis and control using Machine Learning algorithms**" is submitted by me to the Department of Electrical Engineering, Delhi Technological University, Delhi for the partial fulfillment of the requirements for the award of the degree of Master of Technology, and that this submission is original and not copied from any source without proper citation. This work has never been used to give a degree, diploma associate ship, fellowship, or any other equivalent title or recognition.

Place: Delhi

Date: May, 2022

Faiz Mohammad Ali
FAIZ MOHAMMAD ALI

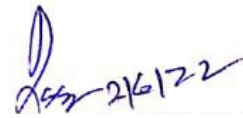
ELECTRICAL ENGINEERING DEPARTMENT
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

CERTIFICATE

I hereby certify that the MAJOR PROJECT titled " **Cart pole system analysis and control using Machine Learning algorithms**" submitted by Faiz Mohammad Ali, 2K20/C&I/01, Electrical Engineering Department, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of a Master of Technology degree is a record of the project work completed by the student under my supervision. To the best of my knowledge, this work has never been submitted in part or in whole for any degree or diploma at this university or anywhere else.

Place: Delhi

Date: May 2022



PROF. MADHUSUDAN SINGH

Electrical Engineering Department

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering Bawana Road, Delhi-110042)

ACKNOWLEDGEMENT

I am really thankful to the Department of Electrical Engineering at Delhi Technological University (DTU) for allowing me to work on this project.

I wish to express my sincere gratitude to my supervisor **Prof. Madhusudan Singh** of the Department of Electrical Engineering at DTU, for his able guidance and support in completing this project work, which has been extremely helpful in carrying out my current work.

Finally, I'd want to thank all of the academic members of DTU's Electrical Engineering Department for their intellectual support throughout my M.Tech studies.

ABSTRACT

The cart and pole system balancing is a classical benchmark problem in control theory which is also referred as inverted pendulum. It is a prototype laboratory model of an unstable mechanical system. It is mainly used to model the control problems of rockets and missiles in the initial stages of their launch.

This system represents an unstable system because an external force is required to keep the pendulum in vertically upright position when cart moves on horizontal track.

Designing optimal controllers for the Cart and pole system is a challenging and complex problem as it is an inherently nonlinear system. The principal advantage of reinforcement learning (RL) is its ability to learn from the interaction with the environment and provide an optimal control strategy. In this project, RL is explored in the context of control of the benchmark cart-pole dynamical system. RL algorithms such as Q-Learning, SARSA, and value-function approximation applied to Q-Learning are implemented in this context. By using a fixed Force value of +10N or -10N, decided by a policy that maximizes the approximate value function, the agent achieves optimal control of the system.

TABLE OF CONTENTS

Acknowledgement	IV
Abstract	V
Chapter 1: Introduction.....	8
1.0 Introduction.....	8
1.1 Literature review.....	10
1.1.1 Introduction to literature review	10
1.1.2 Referred Literature	10
1.2 Motivation for machine learning algorithms.....	13
1.3 Objective of present work.....	14
1.4 Organization of thesis.....	14
Chapter 2: Mathematical modeling of Cart Pole dynamics and applications of Reinforcement Learning algorithms.....	15
2.0 Overview of cart pole dynamics and reinforcement learning	15
2.1 Cart Pole system	16
2.2 Reinforcement Learning	18
2.2.1 Elements of Reinforcement Learning	18
2.2.2 Finite Markov Decision Processes	19
2.2.3 The Agent – Environment Interface	19
2.2.4 Returns	20
2.2.5 The Markov Property	21
2.3 Cart Pole Model for Reinforcement Learning.....	22
2.4 Temporal Difference Learning	24
2.5 Conclusion	26
Chapter 3: Application of Reinforcement Learning for stabilization of cart pole system.....	26
3.1 On-policy Temporal Difference control algorithm (SARSA)	26
3.2 Off-Policy Temporal Difference control algorithm (Q-learning).....	28
3.3 Q-Learning with value function approximation.....	30
Chapter 4: Conclusions and Future scope of work.....	33
References	35
Appendix	41

LIST OF TABLES AND FIGURES

List of tables

TABLE NO.	LABEL
Table – 2.1	Constraints of the Cart-pole environment considered

List of figures

FIGURE NO.	LABEL
Figure – 1.1	Cart-Pole System
Figure - 1.2	Cart-Pole state based on action
Figure – 2.1	Diagram of the pendulum environment with actuating forces
Figure – 2.2	Forces acting on Cart Pole System
Figure – 3.1	The agent – environment interaction in reinforcement learning
Figure – 3.1	SARSA method Cart-Pole state against time steps
Figure – 3.2	Q-Learning method Cart-Pole state against time steps
Figure – 3.3	RBF neural network
Figure – 3.4	Value function approximation method Cart-Pole state against time steps

CHAPTER – 1

INTRODUCTION

1.0 INTRODUCTION

In the context of control system, the cart pole is a benchmark control problem. A cart pole system consists of a pole pivoting on a cart that may travel along a horizontal axis. The main challenge is to use bi-directional forces applied on the cart by an electromechanical system and balance the pole in an upright position.

Fig 1.1 shows a schematic diagram of Cart-pole balancing which is a well-known control benchmark problem. It is designed mechanical system which is inherently unstable and underactuated. The dynamics of this system are used to better comprehend challenges like maintaining balance (e.g., a human walking), self-balancing systems and thruster control of rockets.

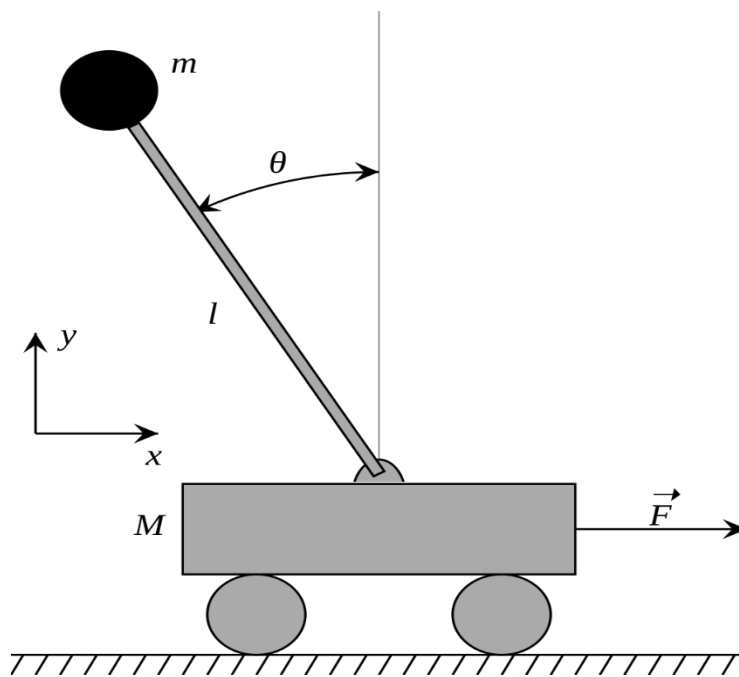


Figure 1.1 – Cart-Pole System

The Cart Pole balancing problem has been used as a benchmark for Reinforcement Learning algorithms for many decades. While the newer definitions of the Cart Pole balancing problem focus on achieving control in complex state and action spaces, the older definitions of the problem are fundamental and have been derived from an Adaptive Control technique known as the BOXES and thus has been explored by both the Reinforcement Learning and Control research communities in the past. In addition to its prominence in these literatures, this problem is a challenge for the RL agent as it has to select and take actions in a very limited and discrete action space.

The condition of the cartpole (if it's tilted to the left or right) is referred to as state. Any data that represents the cart-pole, which includes the cart's speed and position, the pole's angle, and pole speed on the tip, may be taken into consideration as states. For this work the cart's state is the four characteristics listed above.

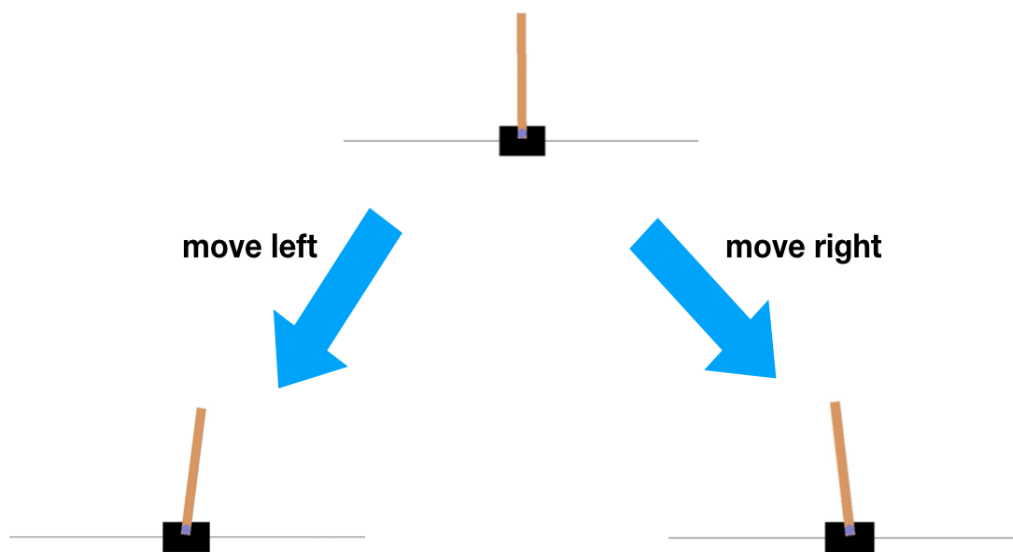


Figure 1.2 – Cart-Pole state based on action

Depending at the motion taken, it may result in one-of-a-kind different states as shown in Fig.1.2. Suppose the pole is beginning straight, if the cart is going left, the pole is more often than not to head right, that's a brand-new state. Therefore, for the duration of every time-step, any movement taken will usually cause a one-of-a-kind state.

1.1 LITERATURE REVIEW

1.1.1 AN OVERVIEW

With increasing demand for complex nonlinear control systems, design of optimal controllers through classical methods is challenging. The fact that Reinforcement Learning enables a system to learn through exploration, and adapt to a dynamic environment makes it optimal to be applied to control a nonlinear complex system. Cart Pole balancing problem is a non-linear control problem, with inherent instability and is often used as a benchmark for control techniques. Cart Pole balancing problem focuses on achieving control in complex state and action spaces. In literature various techniques are used to balance the cart pole system as well as various other systems.

1.1.2 REFERRED LITERATURE

In [1] an artificial neural network is utilised to control the angle and position of a non-linear inverted pendulum system as an artificial intelligence technology. In [2] a continuous-state system is considered and pursue a near-optimal policy through online learning and new online RL algorithm—MSEC (Multi-Samples in Each Cell) is proposed. In [3] linear and nonlinear control approaches are used to solve the problem of balancing an inverted pendulum on an unmanned aerial vehicle (UAV). In [4] an experience replay is proposed for least-squares policy iteration (ERLSPi) for improving the utilization efficiency. The inverted pendulum system is then solved using the ERLSPi approach. A game theoretic Lyapunov fuzzy controller that is both safe and stable is utilised to improve a Markov games-based reinforcement learning controller that is additionally hybridised with a Lyapunov theory-based control in [5].

In [7] a new approach is proposed for stabilization of a cart inverted pendulum system using PID controller based on Linear Quadratic Regulator (LQR) and Artificial Neural Network (ANN). In [8] a method is proposed to research the assistive techniques immediately from interactions among the person and the robotic and system of a learning hassle of assistive techniques as a coverage seek hassle to relieve heavy burdens to the person for information acquisition is executed with the aid of using exploiting a information-green model-primarily based

totally reinforcement learning framework. In [9] one-of-a-kind reinforcement algorithms are used, Q-learning and Adaptive Heuristic Critic (AHC) algorithm, on famous cart-pole balancing hassle and their overall performance is examined. In [10] it's miles proven that an intermittent controller may be installed obviously thru reinforcement learning. In [11] the symmetry definition is elevated from finite to countless state space after which a unique sort of symmetric foundation capabilities is proposed for fee characteristic approximation to combine the earlier expertise of symmetry approximately the surroundings for big or maybe countless state space.

In [12] Reinforcement Learning has been utilized to develop a controller for stabilization of a Double Inverted Pendulum. In [13] a robot version-primarily based reinforcement learning approach that mixes thoughts from version identity and version predictive control is presented.

In [14] a self-learning method is evolved which permits the agent to adaptively expand an inner reward signal primarily totally based on a given ultimate goal, without requiring a specific external reward signal from the environment. The problem of cart-pole balancing withinside the continuous state space setup with restrained track length is provided in [15].

In [16] a Control Algorithm by implementing the Q-Learning techniques in the PD control scheme is presented.

The challenge of stabilising an inverted pendulum system with known/unknown internal dynamics is described in [17] with simulations to show the advantages of the RL technique over the conventional approach. In the method proposed in [18] the learning cycles of online reinforcement learning systems are adaptively changed to acquire a necessary and sufficient set of states for them by using a growing self-organizing map to estimate the state for fast learning speed. To enhance the time of the single inverted pendulum swinging up, [19] uses an improved reinforcement learning algorithm that is modelled in a double-layer BP neural network. The MQTT protocol is used on the Ethernet connection as a deep reinforcement learning environment, and the agent is learned to control the real device located remotely from the controller, and the classical PID controller is also used to implement imitation reinforcement learning and facilitate the learning process in [20]. A rotary inverted pendulum composed of a cyber environment and physical environment based on the OpenFlow network, and the MQTT protocol is used on the Ethernet connection as a deep reinforcement learning environment, and the agent is learned to control the real device A cyber physical system (CPS) is constructed in [21] to show that a deep

RL agent can successfully manage a rotary cartpole system.

In [23] stochastic variations of a classic cartpole balancing task are implemented using three algorithms: Vanilla Policy Gradient, Natural Policy Gradient, and TRPO. In [24] the control problem of nonlinear system under continuous state space is solved using a neural Q-learning algorithm based on residual gradient method.

In [25] DQN based on VREP simulation environment, is used to try to solve inverted pendulum problem. In [26], a successful solution to the pruning performance reduction problem in the DRL domain is presented, as well as a functional approach known as Policy Pruning and Shrinking (PoPS) for training high-performing DRL models while preserving a compact representation of the DNN. In [27] a brand new set of rules primarily based totally on SARSA is proposed to keep away from the overestimation hassle in conventional reinforcement learning. In [28] the train throughput evaluation is executed with RLlib and IMPALA on CartPole and Pong and the results of diverse scalability metrics, clustering, and commentary dimensions on train throughput are analyzed.

In [30] RLHTM a brand new learning technique of Reinforcement learning is provided primarily based totally on brain-stimulated learning called HTM and the overall performance of Hierarchical Temporal Memory (HTM) in a reinforcement learning device is analyzed via way of means of changing the Q characteristic in RL with HTM.

In [31] a federated reinforcement learning based on multi agent environment is proposed. In [32] a lightweight on-device reinforcement learning approach for low-cost FPGA devices is proposed which exploits a recently proposed on-device learning approach based on neural networks that does not employ backpropagation but instead uses the OS-ELM (Online Sequential Extreme Learning Machine) training algorithm. In addition, for on-device reinforcement learning, a combination of L2 regularisation and spectral normalisation is proposed. State change predictions are employed as an unbiased and non-sparse supplement for TD-targets in [34], and transfer learning from model dynamics prediction to Q value function approximation is allowed by training a forward model that shares the initial layers of a Q-network. In [35] learning a potential function concurrently with training an agent using a reinforcement learning algorithm based on the potential-based reward shaping framework, which guarantees policy invariance is proposed. The topic of transfer learning between areas with strong similarities is examined in [36].

Various deep Q learning implementations for discrete action space systems are examined in

[37], and the efficiency of the solutions for the classic Cartpole problem transferred to the Gazebo environment is investigated. A unique model compression framework for DRL models is described in [38], which uses a sparse regularised pruning method and policy shrinking technology to achieve a great balance between high sparsity and compression rate.

[39] investigates the adaptive optimal stationary control of continuous-time linear stochastic systems with both additive and multiplicative noises using reinforcement learning techniques, and proposes a novel off-policy reinforcement learning algorithm based on policy iteration called optimistic least-squares-based policy iteration.

In [40], a twin delayed deep deterministic policy gradient (TD3) actor-critic algorithm is utilised to solve the Inverted Pendulum control issue, and a unique dynamic and enhanced reward function is proposed for the same goal.

1.2 MOTIVATION FOR MACHINE LEARNING ALGORITHMS

The development of computational power is constantly on the rise and makes for new possibilities in a lot of areas. Two of the areas that has made great progress thanks to this development are control theory and artificial intelligence. The most eminent area of artificial intelligence is machine learning. The difference between an environment controlled by control theory and an environment controlled by machine learning is that the machine learning model will adapt in order to achieve a goal while the classic model needs preset parameters. This supposedly makes the machine learning model more optimal for an environment which changes over time. This theory is tested in this research work on a model of an inverted pendulum. Three different machine learning algorithms are implemented on a cart pole model. Changes are made to the model machine learning algorithms are tested. As a result, one of the algorithms were able to mimic the classic model but with different accuracy.

1.3 OBJECTIVE OF PRESENT WORK

This dissertation mainly focuses on applications of machine learning algorithms for balancing of cart pole system. The following are main contributions in this project.

- i. To study the mathematical model of cart pole dynamics and identification of system.
- ii. Implement reinforcement learning algorithms (SARSA and Q-Learning) on a discretized cart pole system.

iii. Implement Q-Learning technique with value function approximation on cart pole system.

1.4 ORGAIZATION OF THESIS

This dissertation contains 4 chapters, Chapter – 1 includes a brief introduction of the present work. Chapter 2 explains, dynamics of cart pole system and Reinforcement learning algorithm. Chapter 3 shows the implementation and results of SARSA, Q-Learning and Q-Learning with value function approximation. In chapter 4 the outcomes of this work has been concluded.

CHAPTER – 2

MATHEMATICAL MODELING OF CART POLE DYNAMICS AND APPLICATIONS OF REINFORCEMENT LEARNING ALGORITHMS

2.0 OVERVIEW OF CART POLE DYNAMICS

The inverted pendulum operates in an environment with the following parameters; A cart that has a mass M . External force F is added at the sides. The pendulum itself has a mass m and is connected to the cart through a rigid massless rod with a length l . The pendulum is rotated from the vertical line by a quantity θ in the counter clockwise direction. There's also a friction force f that works in the opposite direction of the external force and a gravitation constant g . Figure 2.4 describes the environment.

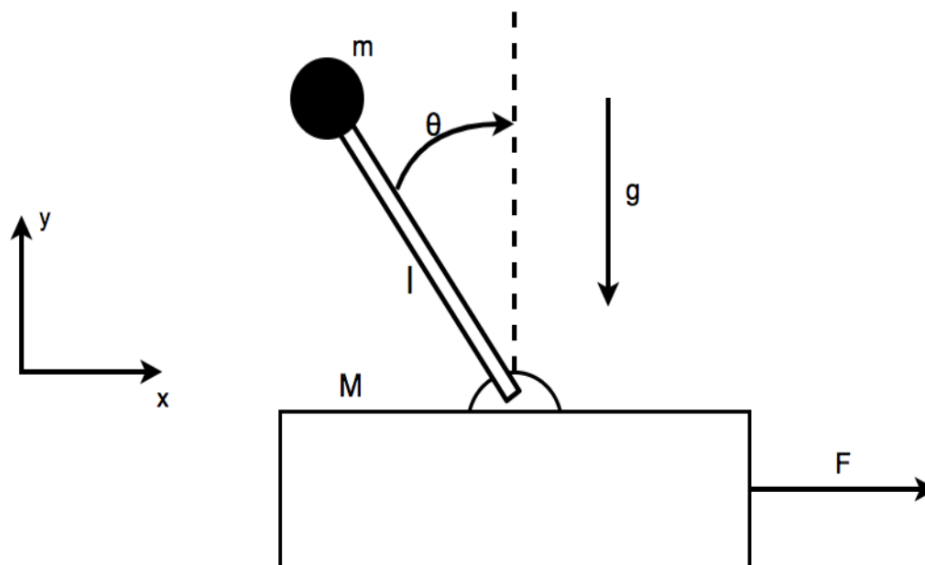


Figure 2.1: Diagram of the pendulum environment with actuating forces

The system state of the pendulum at any time is defined by four state variables:

- i. Angular Position θ
- ii. Angular Velocity $\dot{\theta}$

iii. Linear Position of the cart x

iv. Linear Velocity of the cart \dot{x}

The system has the freedom to move in two different ways, the cart can move horizontal with the x-axis and the pendulum can rotate against its pivot point 360 degrees.

Reinforcement Learning involves learning how to connect situations to behaviours in order to maximise a numerical reward signal. These problems are closed-loop because the learning system's actions influence its later inputs. Furthermore, unlike many forms of machine learning, the learner is not directed specific actions to take, but instead must discover which actions yield the most reward by trying them out. In most complex problems, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. Thus, these are the three most important distinguishing features of a reinforcement learning problem:

- i. Being closed-loop
- ii. Not having commands as to what moves to take, and the effects of those moves consisting of reward signals.
- iii. Operate for longer periods of time

2.1 MATHEMATICAL MODELING OF CART POLE SYSTEM

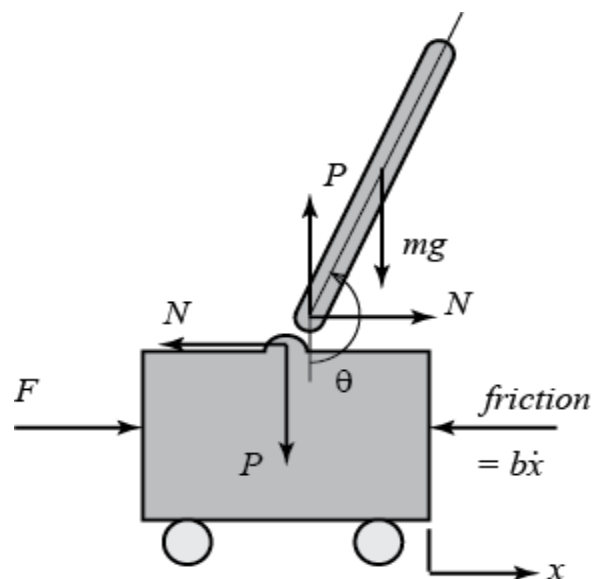


Figure 2.2: Forces acting on Cart Pole System

In section 2.0 it was stated that the pendulum has the freedom to move in two different ways and the forces acting on the pendulum are shown in Fig.2.2.

This leads to two state variables:

x_s = Displacement of cart on x-axis relative to starting position.

θ_s = Angular displacement for the pivot relative to upright position.

To derive the equations of motions using Lagrange's equations (2.1).

$$\frac{d}{dt} \left(\frac{\partial L}{\partial \dot{q}_i} \right) - \frac{\partial L}{\partial q_i} = Q_i \quad (2.1)$$

Where L is the difference between the kinetic energy (T) and the potential energy (V).

$$L = T - V \quad (2.2)$$

The potential energy of the system is going to be the potential energy of the pendulum since the cart will never have any stored energy.

$$V = mgl \cos \theta \quad (2.3)$$

Finding the kinetic energy is a little more complicated since it involves both the pendulum and the cart.

$$\begin{aligned} T &= \frac{1}{2} M \dot{x}^2 + \frac{1}{2} m V_p^2 \\ &= \frac{1}{2} (M \dot{x}^2 + m (V_x^2 + V_y^2)) \\ &= \frac{1}{2} (M \dot{x}^2 + m ((\dot{x} - l \dot{\theta} \cos \theta)^2 + l^2 \dot{\theta}^2 \sin^2 \theta)) \\ &= \frac{1}{2} ((M + m) \dot{x}^2 - m (2 \dot{x} l \dot{\theta} \cos \theta - l^2 \dot{\theta}^2 (\cos^2 \theta + \sin^2 \theta))) \\ &= \frac{1}{2} ((M + m) \dot{x}^2 - m (2 \dot{x} l \dot{\theta} \cos \theta - l^2 \dot{\theta}^2)) \end{aligned} \quad (2.4)$$

By then combining formula 2.2, 2.3, and equation 2.4 can be solved for L.

$$L = \frac{1}{2} ((M + m) \dot{x}^2 - 2m \dot{x} l \dot{\theta} \cos \theta + ml^2 \dot{\theta}^2 - 2mgl \cos \theta) \quad (2.5)$$

Using Lagrange's equations (2.1) and calculating the equations of motions for our state variables x_s and θ_s we can get the equations of motions for the system.

$$x_s : (M + m) \ddot{x} - ml \ddot{\theta} \cos \theta + ml \dot{\theta}^2 \sin \theta = F \quad (2.6)$$

Where F is the external force applied to the state variable x_s

$$\begin{aligned}
\theta_s : m\dot{x}l\dot{\theta}\sin\theta + ml^2\ddot{\theta} - m\ddot{x}l\cos\theta - m\dot{x}l\dot{\theta}\sin\theta - mgl\sin\theta \\
= ml^2\ddot{\theta} - m\ddot{x}l\cos\theta - mgl\sin\theta \\
= l\ddot{\theta} - \ddot{x}\cos\theta - g\sin\theta = 0
\end{aligned} \tag{2.7}$$

Equation 2.7 is equal to 0 because there will be no external force actuating on state variable θ_s .

2.2 REINFORCEMENT LEARNING

Learning from a training set of labelled examples provided by an external supervisor is known as supervised learning. The objective of this kind of learning is for the system to extrapolate or generalize its responses so that it acts correctly in situations not present in the training set. Unsupervised learning deals with finding structure hidden in collections of unlabeled data. Reinforcement learning is therefore, a third machine learning paradigm that deals with a goal-directed agent interacting with an uncertain environment.

One of the challenges that arise in reinforcement learning, and not in other kinds of learning, is the trade-off between exploration and exploitation. The agent must utilise what it already knows to maximise reward, but it must also explore in order to make better action decisions in the future. Neither exploration nor exploitation can be pursued exclusively without failing at the task. In case of a deterministic environment, the agent must explore by trying various actions in each state, and progressively learn to select the best action at each state. On the other hand, a stochastic environment requires repeated trials of the same action at each state in order to obtain an estimate of the amount of reward expected from that state. This dilemma between exploration and exploitation is one of the key features of reinforcement learning.

2.2.1 Elements of Reinforcement Learning

- i. **Policy:** A policy describes the learning agent's behaviour at a specific moment in time, i.e., A policy is the relation of environmental perceptions to steps to perform when such perceptions are realized. It corresponds to the stimulus - response rules of psychology.
- ii. **Reward signal:** In a reinforcement learning problem, the aim is described by means of a reward signal. Any Reinforcement Learning algorithm's sole aim is to maximize the full reward it gets over time. The reward signal as a result defines the agent's effective and terrible events. In a organic system, rewards are analogous to pleasure or pain experiences.

- iii. **Value function:** A state's value function is the full quantity of reward an agent can assume to build up in the future, starting with that state. The RL algorithm estimates the value function as a feature of rewards. Unlike rewards, which decide the immediate, intrinsic desirability of environmental states, values suggest the long-time period desirability of states after accounting for the states which might be in all likelihood to follow and the rewards to be had in those states.
- iv. **Model:** A model simulates the environment's behaviour or, to put it another way, allows conclusions about how the environment will behave. Models are employed in planning, which is a means of deciding on a course of action by anticipating future events.

2.2.2 Finite Markov Decision Processes

Markov Decision Processes (MDPs) are a mathematical framework for describing decision making structure in scenarios where the result is partially random and partially under the decision making agent's control. MDPs are used to tackle a variety of optimization and control problems, which are solved using different reinforcement learning techniques.

2.2.3 The Agent – Environment Interface

The reinforcement learning venture is a easy manner to border the hassle of learning from interaction so one can achieve a goal. The agent is each a learner and a choice maker. The environment refers back to the device with which it interacts, which incorporates the entirety outside of the agent. As visible in Fig.3.1, the agent chooses actions, and the environment responds by providing new conditions to the agent. The environment additionally produces rewards, which can be unique numerical values that the agent tries to maximize over time. A task is a kind of reinforcement learning problem this is described with the aid of using an in-depth description of an environment, inclusive of how rewards are calculated.

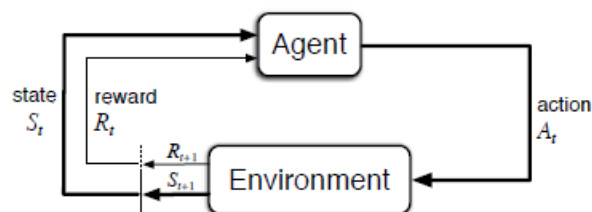


Figure 2.3: The agent – environment interaction in reinforcement learning

The agent and environment interact at different time steps, $t = 0, 1, 2, 3 \dots$. At each and every time step t , the agent receives a new environment state, $S_t \in S$, where S is the set of possible states, and depending on the state chooses an action, $A_t \in A(S_t)$, where $A(S_t)$ is the set of actions possible in state S_t . After one time, the agent receives a numerical reward, $R_{t+1} \in R$ and finds itself in a new state S_{t+1} as a consequence of its action in the previous state. Figure 1 shows the agent – environment interaction.

At all time steps, the agent performs a mapping from states to probabilities of performing each available action. This mapping or state-action pair is the agent's policy denoted as π_t , where $\pi_t(a|s)$ is the probability that $A_t = a$ if $S_t = s$. The agent's policy changes as a result of its experience, as defined by reinforcement learning approaches. The agent's long-term goal is to maximise the total amount of compensation it receives.

This framework is generic and adaptable, allowing it to be used to solve a wide range of issues. The same frame work of MDP has been implemented in the optimization and control problems that have been solved in the later stages of this report.

2.2.4 Returns

The agent's goal is to maximize the cumulative reward it receives in the long run. If the sequence of rewards received after time step t is denoted as $R_{t+1}, R_{t+2}, R_{t+3}, \dots$, then in general, the expected return is maximized over time. If is defined as some specific function of the reward sequence, in the simplest case, the return is the sum of rewards:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \dots + R_T \quad (2.8)$$

where T is a final time step. Each final time step marks the end of an episode, and each episode ends in a special state called the terminal state. The next episode begins from a standard pre-defined starting state, or randomly. These are known as episodic tasks. On the other hand, in many cases the agent-environment interaction does not break naturally into identifiable episodes, but goes on continually without limit. These tasks are called continuing tasks. Various reinforcement learning algorithms have been used to solve the control problem as an episodic task in this report, and their results have been compared.

The additional concept that we need is discounting. According to this method, the agent strives to choose activities that maximise the total of discounted benefits it receives in the future. In particular, it chooses A_t to maximize the expected discounted return:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (2.9)$$

where γ is a parameter, $0 \leq \gamma \leq 1$, called the discount rate.

The discount rate determines the present value of future rewards: a reward received k time steps in the future is worth only γ^{k-1} times what it would be worth if it were received immediately.

2.2.5 The Markov Property

In a reinforcement learning task, a state signal that compactly summarises past feelings while yet retaining all relevant information is desired. Such a signal that succeeds in retaining all relevant information is said to be Markov, or said to have Markov property. Consider how a general environment might respond at time $t+1$ to the action taken at time t . In the broadest sense, this response could be influenced by everything that has happened previously. In this case, the dynamics can be defined only by specifying the complete joint probability distribution:

$$\Pr\{ S_{t+1} = s', R_{t+1} = r \mid S_0, A_0, R_1, \dots, S_{t-1}, A_{t-1}, R_t, S_t, A_t \} \quad (2.10)$$

for all r, s' , and all possible values of the past events: $S_0, A_0, R_1, \dots, S_{t-1}, A_{t-1}, R_t, S_t, A_t$.

A state signal is considered to have the Markov property if the next state and reward earned by the agent from the environment dynamics received at time $t+1$ are solely dependent on the system's state and the RL agent's behaviour at time t . The agent can maintain an estimate of the inherent environment dynamics through State Transition Probabilities, which can be defined by:

$$p(s', r \mid s, a) = \Pr\{ S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a \} \quad (2.11)$$

For all r, s', s and a

If an environment has the Markov property, we may anticipate the next state and predicted next reward based on the current state and action using one-step dynamics. By iterating this equation, one may prove that knowing simply the current state allows one to forecast all future states and expected rewards as well as knowing the entire history up to that point would allow. As a result, Markov states are the finest potential foundation for making decisions. That is, the best policy for selecting actions based on a Markov state is identical to the best policy for selecting actions based on entire histories.

2.2.6 Value Functions

Almost all reinforcement learning methods include predicting value functions, which are functions of states or state-action pairs which assess how good it is for the agent to be in a particular state. The concept of 'how good' is defined in terms of anticipated future rewards, or expected return. The value of a state s under a policy π , denoted as $v_\pi(s)$, is the expected return when starting in s and following π thereafter.

$$v_\pi(s) = E_\pi [G_t | S_t = s] = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right] \quad (2.12)$$

v_π is called the state-value for policy π .

Similarly, $q_\pi(s, a)$ is defined as the value of taking action a in state s under a policy π .

$$q_\pi(s, a) = E_\pi [G_t | S_t = s, A_t = a] = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a \right] \quad (2.13)$$

q_π is called the action-value function for policy π

2.2.7 Bellman Equation

The fact that value functions satisfy specific recursive relationships is a crucial property of reinforcement learning and dynamic programming. For any policy π and any state s , the following consistency condition holds between the value of s and the value of its possible successor states:

$$\begin{aligned} v_\pi(s) &= E_\pi [G_t | S_t = s] = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s \right] \\ &= E_\pi \left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} | S_t = s \right] \\ &= \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) \left[r + \gamma E_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} | S_{t+1} = s' \right] \right] \end{aligned}$$

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s', r} p(s', r | s, a) [r + \gamma v_\pi(s')] \quad (2.14)$$

Equation (2.14) is the Bellman equation for v_π . It expresses a relationship between the value of a state and the values of its successor states. The value function v_π is the unique solution to its Bellman equation. Bellman equation forms the basis of a number of ways to compute, approximate, and learn v_π using various reinforcement learning algorithms.

2.3 CART POLE PLANT MODEL FOR REINFORCEMENT LEARNING

When Reinforcement Learning is applied to the Cart-Pole problem, the dynamics of the Cart-Pole subsystem remain the same. However, a few changes are made in order to incorporate features of Reinforcement Learning and ensure that the Cart-Pole Plant can be completely

described by a Markov Decision Process.

Reinforcement Learning requires that the Environment be in the form that can be described as a Markov Decision Process, where the Action space must be finite. Thus, to limit the number of actions that can be taken on the pendulum, a simplified form is considered and the environment has constraints as given in Table 3.1. Here, a constant magnitude of Force F is considered. As a result, the RL agent can take either a $+F$ or a $-F$ action on the Environment.

Index	Observation	Min	Max
0	Cart Position	-2.4	2.4
1	Cart Velocity	-Inf	Inf
2	Pole Angle	~ -0.209 rad (-12°)	~ 0.209 rad (12°)
3	Pole Angular Velocity	-Inf	Inf

Table 2.1 – Constraints of the Cart-pole environment considered

2.3.1 State Space Quantization

An entire class of Reinforcement Learning algorithms are known as Tabular Lookup Methods. In Tabular Lookup Methods, unlike Continuous methods, the states of the Cart-Pole system are quantized, that is, the quantities pendulum angle, velocity, cart positioning, and velocity across their whole range have been divided into numerous bins. A box is defined here, as a tuple comprising of one bin from each of the four state variables.

These quantized states are the states with which the RL Agent builds a Value function, similar to that of Dynamic Programming. To investigate the influence of quantization on the algorithm's performance, the Cart-Pole plant model was quantized using various quantization parameters.

In the first type of State Space Quantization, the states of the Cart position and velocity as well as the Pole angle and angular velocity have been quantized into 15 bins, and permuted to 162 different boxes representing a tuple of θ , $\dot{\theta}$, x , and \dot{x} using the following rule:

θ : [-12, -6), [-6, -1), [-1, 0), [0, 1), [1, 6), [6, 12] degrees

$\dot{\theta}$: (-inf, -50), [-50,50], [50, inf) degrees /second

x : [-2.4, -0.8), [-0.8, 0.8], (0.8, 2.4] meters

\dot{x} : (-inf, -0.5) [-0.5, 0.5], (0.5, inf) meters/second

2.4 Temporal Difference Learning:

Temporal Difference Learning (TD) is a type of Reinforcement Learning algorithm that uses one-step value function updates and bootstrapping to estimate a state's quality.

TD methods are step-by-step algorithms with online updates of value estimates. At every step of an episode, the quality of the state is updated using the reward obtained at that step and the old estimate of the quality of the next state. In other words, a guess of the state's quality is updated towards a better guess.

The TD Learning update equation for:

i. Prediction:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_t + \gamma V(S_{t+1}) - V(S_t)] \quad (2.15)$$

Where S_t corresponds to the current state,

S_{t+1} corresponds to the next state,

$V(S_t)$ is the quality of the agent being in state (S_t)

(R_t) is the reward obtained by from state (S_t)

ii. Control:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_t + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \quad (2.16)$$

Where (S_t, A_t) corresponds to the current state and action,

(S_{t+1}, A_{t+1}) corresponds to the next state and action,

$Q(S_t, A_t)$ is the quality of the agent being in state S_t and taking action A_t

R_t is the reward obtained by taking an action A_t from state S_t

When using Model Free Learning for Control, there are two ways to join Policy evaluation and Policy improvement into Generalized Policy Iteration. Exploration of the environment is necessary to ensure that the agent is not trapped inside the local maxima of expected return while selecting actions based on the quality of the state. This exploration can be performed in many ways, one being to choose actions randomly in the given state. On the other hand, Exploitation or choosing the action which results in the maximum reward is required to achieve the objective. This trade-off is handled by two methods of Model Free control:

i. On-Policy Control:

The single policy algorithm is utilized by the agent to choose actions from the action value estimates in addition to generating action value assessments. Example SARSA, etc.

ii. Off-Policy Control:

The agent selects actions from action value estimates using one policy algorithm, usually a greedy policy, and generates action value estimates using another policy method, usually an exploratory policy. Example Q-Learning.

2.5 CONCLUSION

System dynamics is an important part of control process. In this chapter a cart pole system and its characteristics were explained. Further, reinforcement learning and its elements were discussed. And pseudo code for SARSA and Q-Learning were also established.

CHAPTER – 3

APPLICATION OF REINFORCEMENT LEARNING FOR STABILIZATION OF CART POLE SYSTEM

3.1 ON-POLICY TEMPORAL DIFFERENCE CONTROL ALGORITHM (SARSA)

When used for control, the On-policy TD algorithm, also known as SARSA (representing State-action, Reinforcement, next State-action), involves update of an action value function $Q(S, A)$ at every step. The SARSA update equation is given by:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha [R_t + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \quad (3.1)$$

Here, $R_t + \gamma Q(S_{t+1}, A_{t+1})$ is the TD target and the expression in rectangular brackets is the TD error. We consider each step in the algorithm to represent a State-action pair:

- i. The current state, S_t or the internal state of the Cart Pole dynamics represented in a form that the RL agent can interpret.
- ii. The action A_t to be taken from that state. Cart Pole balancing action might be either a steady acceleration of the cart towards the LEFT or RIGHT side of the track.

The reinforcement signal, R_t is the reward or punishment that the agent receives after the time step t . As the Objective requires the Pole and the Cart to meet the restrictions, we punish the agent with a Reinforcement of -1 if either the Pole or the Cart does not meet its restrictions. When maintaining the restrictions, no reinforcement is rewarded to the agent.

To create a balance between Exploration and Exploitation, stochastic policies are usually considered to select the action in the given state. But the actions defined by the MDP ensure that the region of the state space corresponding to optimal policy is explored even when a greedy policy is used.

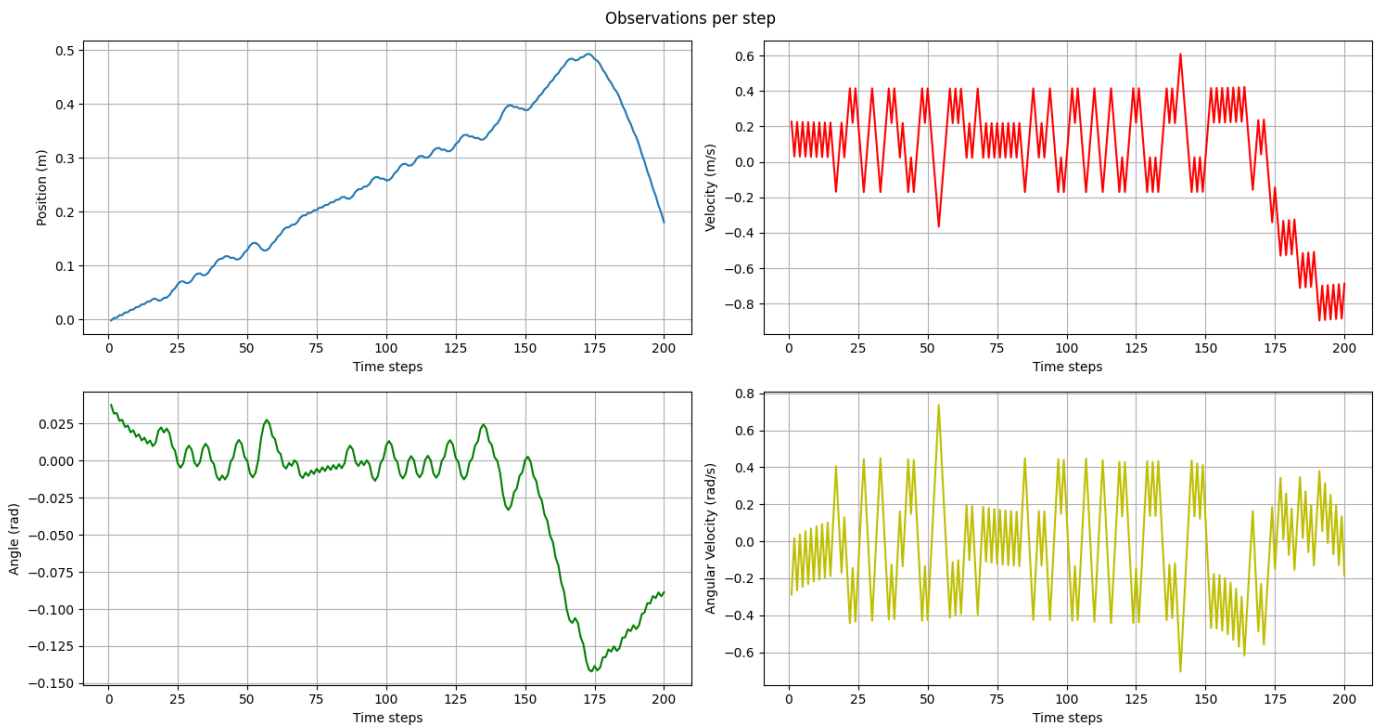
Unlike Q-learning, SARSA — or State-Action-Reward-State-Action — is an on-policy method:

its update is done using the value of the next state and the action of the current policy. The agent follows the current policy and estimates the state-action pairs accordingly, which is the on-policy assumption.

3.1.1 IMPLEMENTATION OF SARSA ALGORITHM

SARSA can be used to solve the Cart Pole balancing problem with the following algorithm:

- Initialize all $Q(S, A) \forall S \in \mathbf{S}, A \in \mathbf{A}$
- For each episode:
 - For each step in an episode:
 - Given current state S_t , choose A_t using $A_t = \operatorname{argmax}_A Q(S_t, A)$
 - Take the action A_t
 - Observe R_t and S_{t+1} from the environment
 - Update the Action value function $Q(S_t, A_t)$, towards the TD target using the SARSA update equation
 - Until the terminal state, where the state S_t exceeds the limitations defined by the Objective.



▪ *Figure 3.1 – SARSA method Cart-Pole state against time steps*

3.1.2 SIMULATION RESULTS

As observed in Fig.3.1 the agent gets a high reward somewhat consistently except for the random time it fails to keep it up. This happened during the testing episode 175 when the agent struggled to keep the pendulum up straight. As a result, the SARSA algorithm only achieves the limited performance. Regardless, the model is still staying within the positive reward domain so all is not completely lost, and the cartpole is kept balanced.

3.2 OFF-POLICY TEMPORAL DIFFERENCE CONTROL ALGORITHM (Q-LEARNING)

Q-Learning is an Off-Policy algorithm since two different policies are utilized by the agent. The policy used to select actions using the state-action values is the greedy policy, given by $\max_{a'} Q(S_{t+1}, a')$. The action-value estimations, on the other hand, are frequently generated using an experimental policy. The Q-learning update equation is given by:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha [R_t + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t)] \quad (3.2)$$

Here, $R_t + \gamma \max_{a'} Q(S_{t+1}, a')$ is the Q-target and the expression in rectangular brackets is the Q-error. The reinforcement given to the Q-learning agent is the same as the reinforcement given to the SARSA agent.

The two policy algorithms used by Off-Policy Control are if greedy policies because the state space of the Cart-Pole MDP is explored even with a greedy policy. This subset of the Off-Policy method converges on an On-Policy approach as a result of this decision.

Through trials-and-errors, a Q-value for each state-action pair is found. The desirability of an action in the current state is represented by this Q-value. If the environment remains static (i.e. the physics or cause-and-effect relationships do not change), the Q-values will converge over time, and the best policy for a given state will be the action with the highest Q-value.

To use Q-Learning, the continuous dimensions have to be discretized to a number of buckets. In general, having fewer buckets and keeping the state-space as compact as feasible is preferable. Training can be done rapidly because there are fewer perfect polices to find. However, obfuscating essential information by discretizing the state-space too coarsely can stymie

convergence.

3.2.1 IMPLEMENTATION OF Q-LEARNING ALGORITHM

Q-learning can be used to solve the Cart Pole balancing problem with the following algorithm:

- Initialize all $Q(S, A) \forall S \in \mathbf{S}, A \in \mathbf{A}$
- For each episode:
 - For each step in an episode:
 - Given current state S_t , choose A_t using $A_t = \operatorname{argmax}_A Q(S_t, A)$
 - Take the action A_t
 - Observe R_t and S_{t+1} from the environment
 - Update the Action value function, $Q(S_t, A_t)$ towards the Q-target $R_t + \gamma \max_{a'} Q(S_{t+1}, a')$ using the update equation
 - Until the terminal state, where the state S_t exceeds the limits set by the objective.

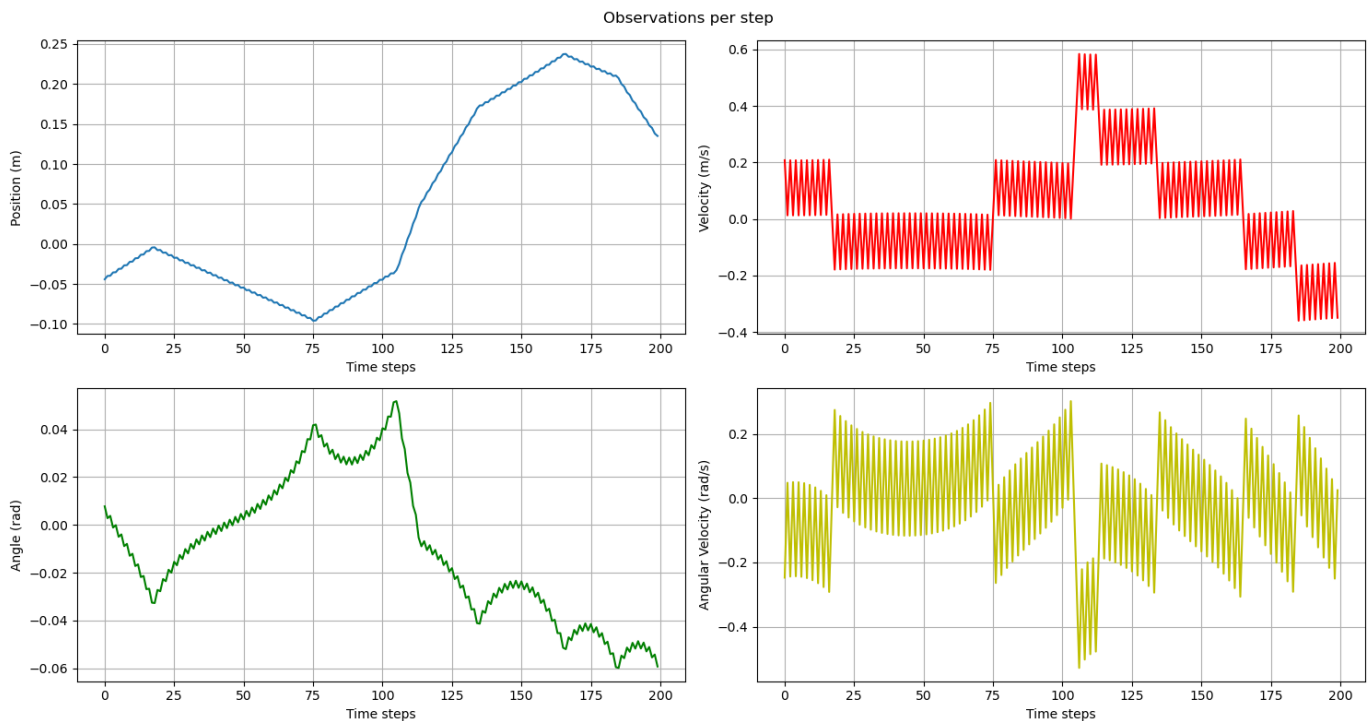


Figure 3.2 – Q-Learning method Cart-Pole state against time steps

3.2.2 SIMULATION RESULTS

Q-Learning using θ , $\dot{\theta}$, x , and \dot{x} is implemented with 3 buckets for $\dot{\theta}$, x , and \dot{x} , and 6 buckets for θ . With this the Cart-Pole problem is solved within 1500 episodes. It was still a bit far from the best solution as observed in Fig.3.2. Upon observation the cart didn't go out of bounds very often. For 200-time steps, the pole is balanced. It typically doesn't drift very far while balancing the pole.

3.3 Q-LEARNING WITH VALUE FUNCTION APPROXIMATION

The results obtained by using the above Reinforcement Learning methods on the Cart-Pole problem have a major drawback. Their assumption of a discretized state space increases the sensitivity of choosing the value of constant force to be applied by the Agent on the cart, and does not accurately represent a real-world scenario of a Cart-Pole system. Also, in the discrete state space, Constant force should be chosen in such a way that the system crosses one box and then reaches another at the conclusion of each time step. In other words, the constant force should cause $P_{ss}^a = 0$ and this increases the difficulty in selection of the constant force value. Thus, a Continuous state space must be considered in order to overcome these drawbacks. The MDP must be updated since the state representation in any Reinforcement Learning issue is determined by the Markov Decision Process. This modified MDP is known as the Continuous State MDP.

Continuous State MDPs represent all the states in terms of a continuous set of values. Thus, values that are assigned to states must be relative to some standard base value. These values relative to the standard base are known as features. In fact, the continuous state of the cart-pole system can be represented by four features. The same features have been used in Plant models, while attempting to stabilize the Cart-Pole system using conventional controllers:

1. Distance of the cart from the center of the track (x):

As the cart is placed on a track system of finite length, and that the objective requires that the cart be positioned within the limits of this track at all times during the experiment, the distance of the cart must be measured and examined by the RL Agent.

2. Angle of the pole with respect to the upright position (θ):

The pole angle must be measured at all times, as the RL Agent must ensure that the pole

remains upright and does not fall towards its stable equilibrium.

3. Velocity of the cart (\dot{x}): Derivative of x
4. Angular velocity of the pole ($\dot{\theta}$): Derivative of θ

These features form the continuous state in the Continuous State MDP. This state value is then said to have a domain of \mathbb{R}^4 , which contains 4 real numbers. As a result, the continuous state $x(s)$ contains infinite number of states. $x(s)$ can be written as:

$$x(s) = \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix} \in \mathbb{R}^4$$

3.3.1 Value Function Approximation

With continuous state MDP, it is not possible to update the value of every state individually, as each state $x(s) \in \mathbb{R}^4$. Also, storing a separate value to represent the quality of each state result in very large MDP, which cannot be stored in the memory efficiently. These drawbacks call for a new value function with the following features:

1. Generalizes values from states visited by the agent, towards states that:
 - a. Have not been visited by the agent
 - b. Belong to the neighborhood of the visited states
2. A set of parameters $w \in \mathbb{R}^4$, which can represent the quality of all states without taking up large memory space.

3.3.2 RBF neural network model

As shown in Figure 6.1, the RBF (Radial Basis Function) neural network is a three-layer forward network. It's three layers are the input, hidden, and output layers. The input layer is connected to the hidden layer by a non-linear connection mapping, and the hidden layer is connected to the output layer by a linear relationship mapping. As shown in Figure.2, x_1, x_2, \dots, x_n are the input of the input layer. h_j is a hidden layer Gauss basis function. w_1, w_2, \dots, w_n are the weights from the hidden layer to the output layer. y_1 is the actual output. The use of RBF is shown in Fig.6.2.

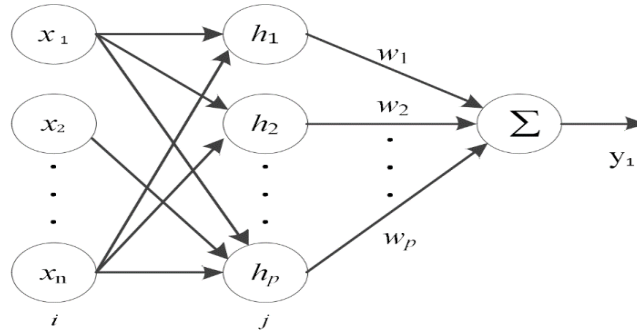


Figure 3.3 - RBF neural network

3.3.3 IMPLEMENTATION OF Q-LEARNING WITH FUNCTION APPROXIMATION

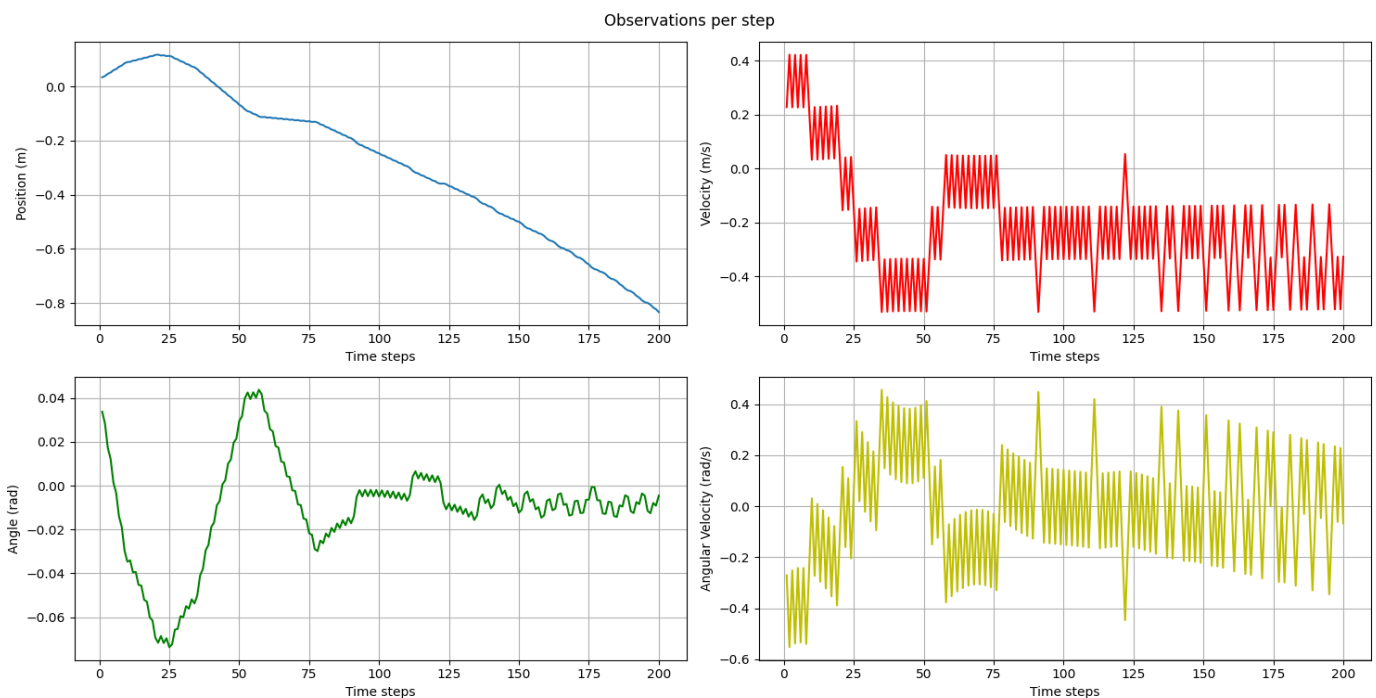


Figure 3.4 – Value function approximation method Cart-Pole state against time steps

3.3.4 SIMULATION RESULTS

Q-Learning using RBF is implemented with 3 buckets for $\hat{\theta}$, \mathbf{x} , and $\dot{\mathbf{x}}$, and 6 buckets for θ . With this the Cart-Pole problem is solved within 1500 episodes. An optimal result is achieved and the pole does not go beyond -3.43° to $+2.29^\circ$ and $[-1.14^\circ, 0^\circ]$ after a stable state is achieved and the cart doesn't go out of bounds very often as seen in Fig.3.4. For 200-time steps, the pole is balanced. It typically doesn't drift very far while balancing the pole.

CHAPTER – 4

CONCLUSIONS AND FUTURE SCOPE OF WORK

4.0 Main Conclusions

A class of machine learning algorithms are implemented for balancing of cart pole system and a comparative study based on the stability of the pole and the time required to stabilize the pole and maintain the obtained stability for longer duration is presented in this thesis. While the result of SARSA(0) achieves optimal policy, a policy for which it achieves the Objective of the Cart-Pole problem but it is unable to maintain the achieved stability for a longer duration of time. The pole is almost dropped during the experiment and the pendulum angle also reaches the edge of its restriction limits.

The pendulum angle oscillates between $\pm 1.4^\circ$ but later shoots up to -8.02° . The cart position is between $+0.5\text{m}$ and $+0.2\text{m}$. This shows the effective performance of the algorithm, although, the Cart still drifts slowly towards the right of the track, which can cause the Cart to reach the track limit if the optimal policy is implemented for a prolonged period.

Due to the convergence of the Off-Policy approach towards the On-Policy SARSA(0) approach, the results of Q-Learning applied to the Cart-Pole problem is similar to the results of the SARSA(0) algorithm applied on the Cart-Pole problem but a superior policy is derived and hence improved results are observed.

The pendulum angle varies from $+2.29^\circ$ to -3.43° in 200 steps. The cart position is between -0.1m and $+0.25\text{m}$. It is again observed that the Cart still drifts slowly towards the right of the track, which can cause the Cart to reach the track limit if the optimal policy obtained here is implemented for a prolonged period.

Value Function Approximation applied on the Cart-Pole problem using Q-Learning, gives near to accurate results. By using a fixed Force value of $+10\text{N}$ or -10N , decided by a policy that

maximizes the approximate value function, the Agent achieves Optimal Control. This algorithm ensures that all oscillations are suppressed to a minimum that are determined by the constant Force values. In the implementation, the Pendulum angle varies from -3.43° to $+2.29^\circ$ and remains within a small range of $[-1.14^\circ, 0^\circ]$ during steady state. Along with this, the Cart position with respect to the center of the track is in the range $[-0.8, 0]$ m. Although there are a few large oscillations at the beginning of the episode of Optimal control these oscillations still lie much within the limits defined by the Cart Pole problem. Similarly, the Cart Position at the end of 200 steps is also within the limits of the Cart-Pole problem. This represents the most optimal control among the three algorithms used and is highly favorable over many conventional control approaches.

4.1 Future Scope of Work

The Cart-Pole Balancing problem is a fundamental problem in Non-linear control systems. As Non-Linear Control has a large number of applications such as Process control, Robotics, Defense and Transportation technology, the Reinforcement Learning algorithm should be further improved to perform complex tasks in all Non-Linear Control applications optimally. To approach complex Non-Linear controls from the Cart-Pole balancing problem, some changes should be made.

From the perspective of the Reinforcement Learning algorithms explored so far, Tabular methods have been the focus and have been compared with a Value function Approximation method which considers a Linear Combination of features. The Value function approximation method has resulted in an optimal policy that converges faster and minimizes the requirement of the actions to be taken multiple times, based on the current states, which is a major disadvantage of the Tabular Methods. With respect to the Cart-Pole balancing problem, the following changes in the RL algorithm can be explored further:

- i. Usage of complex Differentiable Function approximators such as Neural Networks on the problem
- ii. Replacement of the Cart-Pole MDP with complex application specific MDPs.
- iii. Improvement of the RL algorithm in order to deal with Non-linear, Stochastic and Non-stationary environments.
- iv. Selection of hyper-parameters of the RL Agent in order to ensure Optimal policy is reached.

REFERENCES

1. D. Upadhyay, N. Tarun and T. Nayak, "ANN based intelligent controller for inverted pendulum system," 2013 Students Conference on Engineering and Systems (SCES), 2013, pp. 1-6, doi: 10.1109/SCES.2013.6547526.
2. Yuanheng Zhu, Dongbin Zhao and Haibo He, "An high-efficient online reinforcement learning algorithm for continuous-state systems," Proceeding of the 11th World Congress on Intelligent Control and Automation, 2014, pp. 581-586, doi: 10.1109/WCICA.2014.7052778.
3. R. Figueroa, A. Faust, P. Cruz, L. Tapia and R. Fierro, "Reinforcement learning for balancing an inverted pendulum," Proceeding of the 11th World Congress on Intelligent Control and Automation, 2014, pp. 1787-1793, doi: 10.1109/WCICA.2014.7052991.
4. Q. Liu, X. Zhou, F. Zhu, Q. Fu and Y. Fu, "Experience replay for least-squares policy iteration," in IEEE/CAA Journal of Automatica Sinica, vol. 1, no. 3, pp. 274-281, July 2014, doi: 10.1109/JAS.2014.7004685.
5. R. Sharma, "Game theoretic Lyapunov fuzzy control for Inverted Pendulum," 2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions), 2015, pp. 1-6, doi: 10.1109/ICRITO.2015.7359373.
6. Stamenković Lidija J., Antanasijević Davor Z., Ristić Mirjana Đ. et al.. Modeling of methane emissions using artificial neural network approach [J]. Journal of the Serbian Chemical Society, 2015, 80(3).
7. S. D. Hanwate, A. Budhraj and Y. V. Hote, "Improved performance of cart inverted pendulum system using LQR based PID controller and ANN," 2015 IEEE UP Section Conference on Electrical Computer and Electronics (UPCON), 2015, pp. 1-6, doi: 10.1109/UPCON.2015.7456752.
8. M. Hamaya, T. Matsubara, T. Noda, T. Teramae and J. Morimoto, "Learning assistive strategies from a few user-robot interactions: Model-based reinforcement

- learning approach," 2016 IEEE International Conference on Robotics and Automation (ICRA), 2016, pp. 3346-3351, doi: 10.1109/ICRA.2016.7487509.
9. R. Özakar, G. T. Özyer and B. Özyer, "Balancing inverted pendulum using reinforcement algorithms," 2016 24th Signal Processing and Communication Application Conference (SIU), 2016, pp. 1569-1572, doi: 10.1109/SIU.2016.7496053.
 10. K. Michimoto, Y. Suzuki, K. Kiyono, Y. Kobayashi, P. Morasso and T. Nomura, "Reinforcement learning for stabilizing an inverted pendulum naturally leads to intermittent feedback control as in human quiet standing," 2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), 2016, pp. 37-40, doi: 10.1109/EMBC.2016.7590634.
 11. G. -f. Wang, Z. Fang, B. Li and P. Li, "Integrating symmetry of environment by designing special basis functions for value function approximation in reinforcement learning," 2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV), 2016, pp. 1-6, doi: 10.1109/ICARCV.2016.7838691.
 12. S. Raj, "Reinforcement learning based controller for stabilization of Double Inverted Pendulum," 2016 IEEE 1st International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES), 2016, pp. 1-5, doi: 10.1109/ICPEICES.2016.7853147.
 13. C. Xie, S. Patil, T. Moldovan, S. Levine and P. Abbeel, "Model-based reinforcement learning with parametrized physical models and optimism-driven exploration," 2016 IEEE International Conference on Robotics and Automation (ICRA), 2016, pp. 504-511, doi: 10.1109/ICRA.2016.7487172.
 14. H. He and X. Zhong, "Learning Without External Reward [Research Frontier]," in IEEE Computational Intelligence Magazine, vol. 13, no. 3, pp. 48-54, Aug. 2018, doi: 10.1109/MCI.2018.2840727.
 15. S. Panyakaew, P. Inkeaw, J. Bootkrajang and J. Chaijaruwanich, "Least Square Reinforcement Learning for Solving Inverted Pendulum Problem," 2018 3rd International Conference on Computer and Communication Systems (ICCCS),

- 2018, pp. 16-20, doi: 10.1109/CCOMS.2018.8463234.
16. G. Puriel-Gil, W. Yu and H. Sossa, "Reinforcement Learning Compensation based PD Control for Inverted Pendulum," 2018 15th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE), 2018, pp. 1-6, doi: 10.1109/ICEEE.2018.8533946.
 17. A. Sami and A. Y. Memon, "Robust Optimal Control of Continuous Time Linear System using Reinforcement Learning," 2018 Australian & New Zealand Control Conference (ANZCC), 2018, pp. 154-159, doi: 10.1109/ANZCC.2018.8606607.
 18. A. Notsu, K. Yasuda, S. Ubukata and K. Honda, "Optimization of Learning Cycles in Online Reinforcement Learning Systems," 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2018, pp. 3530-3534, doi: 10.1109/SMC.2018.00597.
 19. Y. Chao, L. Yongxin and W. Linglin, "Design of Reinforcement Learning Algorithm for Single Inverted Pendulum Swing Control," 2018 Chinese Automation Congress (CAC), 2018, pp. 1558-1562, doi: 10.1109/CAC.2018.8623253.
 20. J. -B. Kim, H. -K. Lim, C. -M. Kim, M. -S. Kim, Y. -G. Hong and Y. -H. Han, "Imitation Reinforcement Learning-Based Remote Rotary Inverted Pendulum Control in OpenFlow Network," in IEEE Access, vol. 7, pp. 36682-36690, 2019, doi: 10.1109/ACCESS.2019.2905621.
 21. J. -B. Kim, D. -H. Kwon, Y. -G. Hong, H. -K. Lim, M. -S. Kim and Y. -H. Han, "Deep Q-Network Based Rotary Inverted Pendulum System and Its Monitoring on the EdgeX Platform," 2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIIC), 2019, pp. 034-039, doi: 10.1109/ICAIIIC.2019.8668979.
 22. J. Cerviño, J. A. Bazerque, M. Calvo-Fullana and A. Ribeiro, "Meta-Learning through Coupled Optimization in Reproducing Kernel Hilbert Spaces," 2019 American Control Conference (ACC), 2019, pp. 4840-4846, doi: 10.23919/ACC.2019.8814419.
 23. Â. G. Lovatto, T. P. Bueno and L. N. de Barros, "Analyzing the Effect of Stochastic

- Transitions in Policy Gradients in Deep Reinforcement Learning," 2019 8th Brazilian Conference on Intelligent Systems (BRACIS), 2019, pp. 413-418, doi: 10.1109/BRACIS.2019.00079.
24. Y. Si, J. Pu and S. Zang, "Neural Q- Learning Based on Residual Gradient for Nonlinear Control Systems," 2019 International Conference on Control, Automation and Information Sciences (ICCAIS), 2019, pp. 1-5, doi: 10.1109/ICCAIS46528.2019.9074647.
25. X. Li, H. Liu and X. Wang, "Solve the inverted pendulum problem base on DQN algorithm," 2019 Chinese Control And Decision Conference (CCDC), 2019, pp. 5115-5120, doi: 10.1109/CCDC.2019.8833168.
26. D. Livne and K. Cohen, "PoPS: Policy Pruning and Shrinking for Deep Reinforcement Learning," in IEEE Journal of Selected Topics in Signal Processing, vol. 14, no. 4, pp. 789-801, May 2020, doi: 10.1109/JSTSP.2020.2967566.
27. L. Menglin, C. Jing, C. Shaofei and G. Wei, "A New Reinforcement Learning Algorithm Based on Counterfactual Experience Replay," 2020 39th Chinese Control Conference (CCC), 2020, pp. 1994-2001, doi: 10.23919/CCC50068.2020.9189606.
28. S. Jang and N. -S. Park, "Train Throughput Analysis of Distributed Reinforcement Learning," 2020 International Conference on Information and Communication Technology Convergence (ICTC), 2020, pp. 1189-1192, doi: 10.1109/ICTC49870.2020.9289179.
29. M. A. S. Araújo, L. P. C. Alves, C. A. G. Madeira and M. M. Nóbrega, "URNAI: A Multi-Game Toolkit for Experimenting Deep Reinforcement Learning Algorithms," 2020 19th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames), 2020, pp. 178-187, doi: 10.1109/SBGames51465.2020.00032.
30. T. Y. Koffi, C. Tao, T. M. Epalle and B. Mensa-Bonsu, "A Novel Reinforcement Learning Algorithm Based on Hierarchical Memory," 2020 International Conference on Internet of Things and Intelligent Applications (ITIA), 2020, pp. 1-

- 5, doi: 10.1109/ITIA50152.2020.9312239.
31. H. -K. Lim, J. -B. Kim, I. Ullah, J. -S. Heo and Y. -H. Han, "Federated Reinforcement Learning Acceleration Method for Precise Control of Multiple Devices," in *IEEE Access*, vol. 9, pp. 76296-76306, 2021, doi: 10.1109/ACCESS.2021.3083087.
 32. H. Watanabe, M. Tsukada and H. Matsutani, "An FPGA-Based On-Device Reinforcement Learning Approach using Online Sequential Learning," 2021 *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2021, pp. 96-103, doi: 10.1109/IPDPSW52791.2021.00022.
 33. V. Abdelzad, J. Lee, S. Sedwards, S. Soltani and K. Czarnecki, "Non-divergent Imitation for Verification of Complex Learned Controllers," 2021 *International Joint Conference on Neural Networks (IJCNN)*, 2021, pp. 1-8, doi: 10.1109/IJCNN52387.2021.9533410.
 34. A. Tercan and C. W. Anderson, "Increased Reinforcement Learning Performance through Transfer of Representation Learned by State Prediction Model," 2021 *International Joint Conference on Neural Networks (IJCNN)*, 2021, pp. 1-8, doi: 10.1109/IJCNN52387.2021.9533751.
 35. Y. Chen, H. Kasaei, L. Schomaker and M. Wiering, "Reinforcement Learning with Potential Functions Trained to Discriminate Good and Bad States," 2021 *International Joint Conference on Neural Networks (IJCNN)*, 2021, pp. 1-7, doi: 10.1109/IJCNN52387.2021.9533682.
 36. M. Muller-Brockhausen, M. Preuss and A. Plaat, "Procedural Content Generation: Better Benchmarks for Transfer Reinforcement Learning," 2021 *IEEE Conference on Games (CoG)*, 2021, pp. 01-08, doi: 10.1109/CoG52621.2021.9619000.
 37. D. Kozlov, "Comparison of Reinforcement Learning Algorithms for Motion Control of an Autonomous Robot in Gazebo Simulator," 2021 *International Conference on Information Technology and Nanotechnology (ITNT)*, 2021, pp. 1-5, doi: 10.1109/ITNT52450.2021.9649145.
 38. W. Su, Z. Li, Z. Yang and J. Lu, "Deep Reinforcement Learning with Sparse Regularized Pruning and Compressing," 2021 *China Automation Congress*

(CAC), 2021, pp. 8041-8046, doi: 10.1109/CAC53003.2021.9727767.

39. B. Pang and Z. -P. Jiang, "Reinforcement Learning for Adaptive Optimal Stationary Control of Linear Stochastic Systems," in IEEE Transactions on Automatic Control, doi: 10.1109/TAC.2022.3172250.
40. M. Shil and G. N. Pillai, "Inverted Pendulum Control using Twin Delayed Deep Deterministic Policy Gradient with a Novel Reward Function," 2022 IEEE Delhi Section Conference (DELCON), 2022, pp. 1-6, doi: 10.1109/DELCON54057.2022.9752797.

APPENDIX I

Python Code for SARSA

```

import gym
import numpy as np
import math
import matplotlib.pyplot as plt

class CartPoleAgent():
    def __init__(self, buckets=(3,3,3,6), num_episodes=1500, min_lr=0.1, min_epsilon=0.1,
                 discount=0.98, decay=25):
        self.buckets = buckets
        self.num_episodes = num_episodes
        self.min_lr = min_lr
        self.min_epsilon = min_epsilon
        self.discount = discount
        self.decay = decay

        self.env = gym.make('CartPole-v0')

        # [position, velocity, angle, angular velocity]
        self.upper_bounds = [self.env.observation_space.high[0], 0.5,
                             self.env.observation_space.high[2],
                             math.radians(50) / 1.]
        self.lower_bounds = [self.env.observation_space.low[0], -0.5,
                             self.env.observation_space.low[2],
                             -math.radians(50) / 1.]

        self.sarsa_table = np.zeros(self.buckets + (self.env.action_space.n,))

    def discretize_state(self, obs):
        discretized = list()
        for i in range(len(obs)):
            scaling = (obs[i] + abs(self.lower_bounds[i])) / (self.upper_bounds[i] -
                                                             self.lower_bounds[i])
            new_obs = int(round((self.buckets[i] - 1) * scaling))
            new_obs = min(self.buckets[i] - 1, max(0, new_obs))
            discretized.append(new_obs)
        return tuple(discretized)

    def choose_action(self, state):
        if (np.random.random() < self.epsilon):
            return self.env.action_space.sample()
        else:
            return np.argmax(self.sarsa_table[state])

    def update_sarsa(self, state, action, reward, new_state, new_action):
        self.sarsa_table[state][action] += self.learning_rate * (
            reward + self.discount * (self.sarsa_table[new_state][new_action]) -
            self.sarsa_table[state][
                action])

    def get_epsilon(self, t):
        return max(self.min_epsilon, min(1., 1. - math.log10((t + 1) / self.decay)))

    def get_learning_rate(self, t):
        return max(self.min_lr, min(1., 1. - math.log10((t + 1) / self.decay)))

    def train(self):
        for e in range(self.num_episodes):
            current_state = self.discretize_state(self.env.reset())

            self.learning_rate = self.get_learning_rate(e)
            self.epsilon = self.get_epsilon(e)
            done = False

            while not done:
                action = self.choose_action(current_state)
                obs, reward, done, _ = self.env.step(action)
                new_state = self.discretize_state(obs)
                new_action = self.choose_action(new_state)
                self.update_sarsa(current_state, action, reward, new_state, new_action)
                current_state = new_state

```

```

print('Finished training!')

def run(self):
    for it in range(1500):
        self.env.render()
        t = 0
        done = False
        current_state = self.discretize_state(self.env.reset())

        position_list = []
        velocity_list = []
        angle_list = []
        angular_velocity_list = []
        steps = []
        total_rounds = 0
        round = 0
        episode_reward = 0
        while not done:
            self.env.render()
            t = t + 1
            action = self.choose_action(current_state)
            obs, reward, done, _ = self.env.step(action)
            new_state = self.discretize_state(obs)
            current_state = new_state

            round += 1
            position_list.append(obs[0])
            velocity_list.append(obs[1])
            angle_list.append(obs[2])
            angular_velocity_list.append(obs[3])
            steps.append(round)
            total_rounds += round

        if done:
            if round > 150:
                print(f'Threshold reached after {round + 1} iterations.')
            if round < 199:
                position_list = []
                velocity_list = []
                angle_list = []
                angular_velocity_list = []
                steps = []
            else:
                position = position_list
                velocity = velocity_list
                angle = angle_list
                angular_velocity = angular_velocity_list
                time = steps
                plot = True
            break

    if plot:
        fig, ax = plt.subplots(2,2, figsize=(15,8))
        ax[0][0].plot(time, position)
        ax[0][0].set_xlabel('Time steps')
        ax[0][0].set_ylabel('Position (m)')
        ax[0][0].grid()

        ax[0][1].plot(time, velocity, 'r')
        ax[0][1].set_xlabel('Time steps')
        ax[0][1].set_ylabel('Velocity (m/s)')
        ax[0][1].grid()

        ax[1][0].plot(time, angle, 'g')
        ax[1][0].set_xlabel('Time steps')
        ax[1][0].set_ylabel('Angle (rad)')
        ax[1][0].grid()

        ax[1][1].plot(time, angular_velocity, 'y')
        ax[1][1].set_xlabel('Time steps')
        ax[1][1].set_ylabel('Angular Velocity (rad/s)')
        ax[1][1].grid()

        plt.suptitle('Observations per step')
        plt.tight_layout()

```

```

plt.show()
return t

```

Python Code for Q-Learning

```

import numpy as np
import math
import matplotlib.pyplot as plt
import gym

class Agent:
    '''A class to define an agent learning to control the system'''

    def __init__(self, environment, buckets=(3,3,3,6), alpha=0.1, gamma=0.98):
        if not (0 < gamma <= 1):
            raise ValueError("Must be 0 < γ <= 1")
        self.alpha = alpha
        self.gamma = gamma
        self.environment = environment
        self.buckets = buckets
        self.upper_bounds = [self.environment.observation_space.high[0], 0.5,
                             self.environment.observation_space.high[2], math.radians(50)]
        self.lower_bounds = [self.environment.observation_space.low[0], -0.5,
                             self.environment.observation_space.low[2],
                             -math.radians(50)]
        self.Q = np.zeros(self.buckets + (self.environment.action_space.n,))
        self.state = (0, 0, 0, 0)
        self.total_reward = 0

    def discretize(self, observations):
        '''Discretize observations based on the buckets'''
        discretized = []
        for i in range(len(observations)):
            scaling = ((observations[i] + np.abs(self.lower_bounds[i])) / (self.upper_bounds[i]
                - self.lower_bounds[i]))
            scaled_observations = int(round((self.buckets[i] - 1) * scaling))
            scaled_observations = min(self.buckets[i] - 1, max(0, scaled_observations))
            discretized.append(scaled_observations)
        return tuple(discretized)

    def get_reward(self, action, state, reward):
        '''Update Q-value according to the state action pair'''
        self.total_reward += reward
        self.Q[self.state][action] = self.Q[self.state][action] + self.alpha * (
            reward + self.gamma * np.max(self.Q[state]) - self.Q[self.state][action])
        self.state = state

    def choice(self):
        '''Randomly select among the two actions'''
        random_action = self.environment.action_space.sample()
        return random_action

    def greedy_action(self):
        '''Select action that has returned maximum reward'''
        return np.argmax(self.Q[self.state])

def run_experiment(epsilon=1, rounds=500, episodes=1500):
    '''Perform an experiment. Make the agent balance the pole'''
    env = gym.make('CartPole-v0')
    agent = Agent(env)
    # get environment
    obs = env.reset()

```

```

plot = False
steps_per_round = []
for episode in range(epochs):
    position_list = []
    velocity_list = []
    angle_list = []
    angular_velocity_list = []
    steps = []
    total_rounds = 0
    for round in range(rounds):
        env.render()
        p = np.random.random()
        if p < epsilon:
            action = agent.choice()
        else:
            action = agent.greedy_action()
        # apply action
        obs, reward, done, _ = env.step(action)
        state = agent.discretize(obs)
        agent.get_reward(action, state, reward)

        position_list.append(obs[0])
        velocity_list.append(obs[1])
        angle_list.append(obs[2])
        angular_velocity_list.append(obs[3])
        steps.append(round)
        total_rounds += round

    if done:
        if round > 150:
            print(f'Threshold reached after {round + 1} iterations.')
        if round < 199:
            position_list = []
            velocity_list = []
            angle_list = []
            angular_velocity_list = []
            steps = []
        else:
            position = position_list
            velocity = velocity_list
            angle = angle_list
            angular_velocity = angular_velocity_list
            time = steps
            plot = True
            break

    epsilon = epsilon - 0.01
    if epsilon < 0.01:
        epsilon = 0.01

    env.reset()

    steps_per_round.append(round)

env.close()

if plot:
    fig, ax = plt.subplots(2, 2, figsize=(15, 8))
    ax[0][0].plot(time, position)
    ax[0][0].set_xlabel('Time steps')
    ax[0][0].set_ylabel('Position (m)')
    ax[0][0].grid()

    ax[0][1].plot(time, velocity, 'r')
    ax[0][1].set_xlabel('Time steps')
    ax[0][1].set_ylabel('Velocity (m/s)')

```

```

ax[0][1].grid()

ax[1][0].plot(time, angle, 'g')
ax[1][0].set_xlabel('Time steps')
ax[1][0].set_ylabel('Angle (rad)')
ax[1][0].grid()

ax[1][1].plot(time, angular_velocity, 'y')
ax[1][1].set_xlabel('Time steps')
ax[1][1].set_ylabel('Angular Velocity (rad/s)')
ax[1][1].grid()

plt.suptitle('Observations per step')
plt.tight_layout()
plt.show()

fig, ax = plt.subplots(1, 1, figsize=(15, 8))
ax.plot(np.arange(0, episodes), steps_per_round)
ax.set_xlabel('Episodes')
ax.set_ylabel('Iterations')
plt.suptitle('Total iterations per episode')
plt.show()

print("After {} episodes the average cart steps before done was {}".format(episodes,
np.mean(steps_per_round)))

```

Python Code for Q-Learning with Value Function Approximation

```

import gym
import numpy as np
import matplotlib.pyplot as plt
from sklearn.kernel_approximation import RBFSampler

GAMMA = 0.99
ALPHA = 0.1

def epsilon_greedy(model, s, eps=0.1):
    # we'll use epsilon-soft to ensure all states are visited
    # what happens if you don't do this? i.e. eps=0
    p = np.random.random()
    if p < (1 - eps):
        values = model.predict_all_actions(s)
        return np.argmax(values)
    else:
        return model.env.action_space.sample()

def gather_samples(env, n_episodes=10000):
    samples = []
    for _ in range(n_episodes):
        s = env.reset()
        done = False
        while not done:
            a = env.action_space.sample()
            sa = np.concatenate((s, [a]))
            samples.append(sa)

            s, r, done, info = env.step(a)
    return samples

class Model:
    def __init__(self, env):

```

```

    # fit the featurizer to data
    self.env = env
    samples = gather_samples(env)
    self.featurizer = RBFsampler()
    self.featurizer.fit(samples)
    dims = self.featurizer.n_components

    # initialize linear model weights
    self.w = np.zeros(dims)

def predict(self, s, a):
    sa = np.concatenate((s, [a]))
    x = self.featurizer.transform([sa])[0]
    return x @ self.w

def predict_all_actions(self, s):
    return [self.predict(s, a) for a in range(self.env.action_space.n)]

def grad(self, s, a):
    sa = np.concatenate((s, [a]))
    x = self.featurizer.transform([sa])[0]
    return x

def test_agent(model, env, n_episodes=20):
    reward_per_episode = np.zeros(n_episodes)
    for it in range(n_episodes):

        position_list = []
        velocity_list = []
        angle_list = []
        angular_velocity_list = []
        steps = []
        total_rounds = 0
        round = 0
        done = False
        episode_reward = 0
        s = env.reset()
        while not done:

            a = epsilon_greedy(model, s, eps=0)
            s, r, done, info = env.step(a)

            round += 1
            position_list.append(s[0])
            velocity_list.append(s[1])
            angle_list.append(s[2])
            angular_velocity_list.append(s[3])
            steps.append(round)
            total_rounds += round

            episode_reward += r

        if done:
            if round > 150:
                print(f'Threshold reached after {round + 1} iterations.')
            if round < 199:
                position_list = []
                velocity_list = []
                angle_list = []
                angular_velocity_list = []
                steps = []
            else:
                position = position_list
                velocity = velocity_list

```

```

        angle = angle_list
        angular_velocity = angular_velocity_list
        time = steps
        plot = True
    break

    reward_per_episode[it] = episode_reward

if plot:
    fig, ax = plt.subplots(2,2, figsize=(15,8))
    ax[0][0].plot(time, position)
    ax[0][0].set_xlabel('Time steps')
    ax[0][0].set_ylabel('Position (m)')
    ax[0][0].grid()

    ax[0][1].plot(time, velocity, 'r')
    ax[0][1].set_xlabel('Time steps')
    ax[0][1].set_ylabel('Velocity (m/s)')
    ax[0][1].grid()

    ax[1][0].plot(time, angle, 'g')
    ax[1][0].set_xlabel('Time steps')
    ax[1][0].set_ylabel('Angle (rad)')
    ax[1][0].grid()

    ax[1][1].plot(time, angular_velocity, 'y')
    ax[1][1].set_xlabel('Time steps')
    ax[1][1].set_ylabel('Angular Velocity (rad/s)')
    ax[1][1].grid()

    plt.suptitle('Observations per step')
    plt.tight_layout()
    plt.show()

return np.mean(reward_per_episode)

def watch_agent(model, env, eps):
    done = False
    episode_reward = 0
    s = env.reset()
    while not done:
        a = epsilon_greedy(model, s, eps=eps)
        s, r, done, info = env.step(a)
        env.render()
        episode_reward += r
    print("Episode reward:", episode_reward)

if __name__ == '__main__':
    # instantiate environment
    env = gym.make("CartPole-v0")
    model = Model(env)
    reward_per_episode = []
    # watch untrained agent
    watch_agent(model, env, eps=0)

    # repeat until convergence
    n_episodes = 1500
    for it in range(n_episodes):
        s = env.reset()
        episode_reward = 0
        done = False
        while not done:
            a = epsilon_greedy(model, s)

```

```
s2, r, done, info = env.step(a)
# get the target
if done:
    target = r
else:
    values = model.predict_all_actions(s2)
    target = r + GAMMA * np.max(values)
# update the model
g = model.grad(s, a)
err = target - model.predict(s, a)
model.w += ALPHA * err * g
# accumulate reward
episode_reward += r
# update state
s = s2

if (it + 1) % 50 == 0:
    print(f"Episode: {it + 1}, Reward: {episode_reward}")

# early exit
if it > 20 and np.mean(reward_per_episode[-20:]) == 200:
    print("Early exit")
    break

reward_per_episode.append(episode_reward)

# test trained agent
test_reward = test_agent(model, env)
print(f"Average test reward: {test_reward}")

plt.plot(reward_per_episode)
plt.title("Reward per episode")
plt.show()
```


LIST OF PUBLICATIONS

Sr. No	Heading of the paper	Publications
1.	Approaches to EV Charging Structure Planning Using Machine Learning: A Review	3rd International Conference on Climate Change

faiz report

ORIGINALITY REPORT

14%

SIMILARITY INDEX

8%

INTERNET SOURCES

11%

PUBLICATIONS

3%

STUDENT PAPERS

PRIMARY SOURCES

1	arxiv.org Internet Source	4%
2	Savinay Nagendra, Nikhil Podila, Rashmi Ugarakhod, Koshy George. "Comparison of reinforcement learning algorithms applied to the cart-pole problem", 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2017 Publication	1%
3	Submitted to Vrije Universiteit Brussel Student Paper	1%
4	medium.com Internet Source	1%
5	Camilo Andrés Manrique Escobar, Carmine Maria Pappalardo, Domenico Guida. "A Parametric Study of a Deep Reinforcement Learning Control System Applied to the Swing-Up Problem of the Cart-Pole", Applied Sciences, 2020 Publication	<1%

6	www.cl.cam.ac.uk Internet Source	<1 %
7	KARR, Charles L., and Lakhmi C. JAIN. "Cases in geno-fuzzy control", Advances in Fuzzy Systems - Applications and Theory, 1997. Publication	<1 %
8	Submitted to University of Computer Studies Student Paper	<1 %
9	Submitted to Manchester Metropolitan University Student Paper	<1 %
10	Wenyang He, Wenlong Zhao, Yuan Jiang. "Application of Q-Learning and RBF Network in Chinese Chess Game System", IOP Conference Series: Materials Science and Engineering, 2019 Publication	<1 %
11	Submitted to Instituto Tecnológico de Aeronautica (Brazilian Aeronautical Comision) Student Paper	<1 %
12	fugumt.com Internet Source	<1 %
13	Akira Notsu, Koji Yasuda, Seiki Ubukata, Katsuhiro Honda. "Optimization of Learning Cycles in Online Reinforcement Learning Systems", 2018 IEEE International Conference	<1 %

on Systems, Man, and Cybernetics (SMC),
2018

Publication

14	Manuel Lopez-Martin, Belen Carro, Antonio Sanchez-Esguevillas. "Application of deep reinforcement learning to intrusion detection for supervised problems", Expert Systems with Applications, 2020	<1 %
Publication		
15	Submitted to University of Surrey	<1 %
Student Paper		
16	Lecture Notes in Computer Science, 2007.	<1 %
Publication		
17	Submitted to Munich International School	<1 %
Student Paper		
18	Zoltán Somogyi. "The Application of Artificial Intelligence", Springer Science and Business Media LLC, 2021	<1 %
Publication		
19	Mauro Montenegro, Roberto López, Rolando Menchaca-Méndez, Emanuel Becerra, Ricardo Menchaca-Méndez. "Chapter 18 A Parallel Rollout Algorithm for Wildfire Suppression", Springer Science and Business Media LLC, 2020	<1 %
Publication		