International Journal of Science and Research Archive
eISSN: 2582-8185
Cross Ref DOI: 10.30574/ijsra
Journal homepage: https://ijsra.net/

(RESEARCH ARTICLE)

Check for updates

Optimizing reinforcement learning in complex environments using neural networks

Md Nurul Raihen * and Jason Tran

Department of Mathematics and Computer Science, Fontbonne University, Saint Louis, MO, USA.

International Journal of Science and Research Archive, 2024, 12(02), 2047–2062

Publication history: Received on 03 July 2024; revised on 11 August 2024; accepted on 13 August 2024

Article DOI: https://doi.org/10.30574/ijsra.2024.12.2.1471

Abstract

This paper presents the distinct mechanisms and applications of traditional Q-learning (QL) and Deep Q-learning (DQL) within the realm of reinforcement learning (RL). Traditional Q-learning (QL) utilizes the Bellman equation to update Q-values stored in a Q-table, making it suitable for simple environments. However, its scalability is limited due to the exponential growth of state-action pairs in complex environments. Deep Q-learning (DQL) addresses this limitation by using neural networks to approximate Q-values, thus eliminating the need for a Q-table, and enabling efficient handling of complex environments. The neural network (NN), acting as the agent's decision-making brain, learns to predict Q-values through training, adjusting its weights based on received rewards. The study highlights the importance of well-calibrated reward systems in reinforcement learning (RL). Proper reward structures guide the agent towards desired behaviors while minimizing unintended actions. By running multiple environments simultaneously, the training process is accelerated, allowing the agent to gather diverse experiences and improve its performance efficiently. Comparative analysis of training models demonstrates that a well-balanced reward system results in more consistent and effective learning. The findings underscore the necessity of careful design in reinforcement learning systems to ensure optimal agent behavior and efficient learning outcomes in both simple and complex environments. Through this research, we gain valuable insights into the application of Q-learning (QL) and Deep Q-learning (DQL), enhancing our understanding of how agents learn and adapt to their environments.

Keywords: Reinforcement Learning; Neural Network; Q-Table; Optimization; Action and Reward.

1. Introduction

For Machine learning involves extracting knowledge from data in order to make predictions and/or judgments. Machine learning aims to answer two essential issues. First, how can computer systems be built to automatically improve their performance as they gain experience? This entails creating systems that can learn and change over time without human involvement. Second, what are the fundamental principles of statistical computation and information theory that govern learning processes across all systems, whether human, computer-based, or organizational? According to Li, Y. et al. (2017) [1], machine learning research is critical not only for investigating core technical and scientific concerns, but also for developing and using useful software in a variety of industries.

Reinforcement learning (RL) is a multidisciplinary field of study that includes machine learning and optimal control. It focuses on establishing an intelligent agent's optimal behavior in dynamic contexts in order to maximize acquired rewards. RL is a core paradigm in machine learning, alongside supervised and unsupervised learning as fundamental methodologies [See Wikipedia] [2]. Reinforcement learning differs from supervised learning, which uses labeled input/output pairs and immediately corrects errors. Reinforcement learning, on the other hand, learns by trial and error rather than explicit directions. It focuses on optimizing long-term benefits, which may include partial or delayed input. This process requires a mix of exploration (new possibilities) and exploitation (established techniques), according to Raihen et al. (2024) [3, 4, 5, 6].

* Corresponding author: Md Nurul Raihen

Deep reinforcement learning (DRL) combines deep learning with reinforcement learning (RL). Previously out of reach for computers, this field of study has been able to address a wide range of tough decision-making responsibilities. Deep RL thus offers up numerous new applications in domains like as healthcare, robotics, smart grids, finance, and many more. Several works describe how to evaluate RL algorithms. In their 2016 article, Duan et al. benchmarked a number of RL algorithms and made them available as public baseline implementations. Whiteson et al. (2011) [7] propose measures for RL evaluation that can be used broadly. Machado et al. (2017) [8] revisit the Arcade Learning Environment and propose new evaluation methods for these requirements. Nonetheless, this appears to be the first study to address the critical issue of reproducibility and good experimental practice in the context of deep RL, despite previous research in related domains (Raihen 2023 [9]; Boulesteix, Lauer, and Eugster 2013 [10]; Stodden, Leisch, and Peng 2014 [11]; Bouckaert and Frank 2004 [12]; Bouckaert 2004 [13]; Vaughan and Wawerla 2012 [14]).

Reinforcement learning is driven by algorithms that learn and make decisions based on outcomes. After each action, the algorithm evaluates whether the result was positive, neutral, or negative, adapting its strategy accordingly. This approach is particularly effective for automated systems that require making numerous small decisions independently of human oversight. Reinforcement learning is a type of autonomous system that learns through a trial-and-error approach. It engages in actions with the objective of maximizing rewards, effectively learning from its experiences to optimize outcomes.

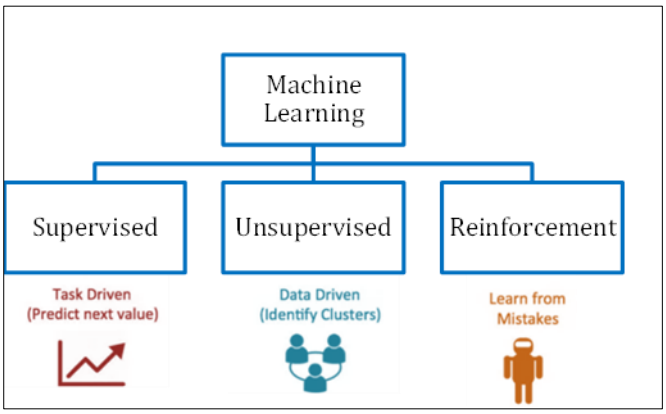


Figure 1 Comparing communication mechanisms: supervised, unsupervised, and reinforcement Learning**.**

In **Fig 1**, when mapping of inputs to outputs is a feature of both supervised and reinforcement learning, their feedback mechanisms are very different. Correct actions required to complete a task make up the feedback in supervised learning. Reinforcement learning, on the other hand, signals positive and negative actions with rewards and penalties, accordingly.

Reinforcement learning originated in the early days of cybernetics and has been inspired by subjects such as statistics, psychology, neurology, and computer science. In recent years, it has received a lot of interest from the machine learning and artificial intelligence sectors. The attractiveness of reinforcement learning stems from its capacity to program agents through a system of rewards and punishments, allowing them to learn tasks without explicit instructions on how to complete them. However, the approach presents significant computational hurdles that must be addressed in order to fully realize its promise, as mentioned by Littman M. L. et al. (1996) [15].

Our study advances the state-of- the-art in model-based reinforcement learning by introducing a system, to our knowledge the first to effectively handle a range of difficult games in the LE (Learning Environment) benchmark. In order to this aim, we investigate numerous stochastic game prediction methods including a novel approach based on 1000 to 100,000 episodes. We propose a Deep Neural Network (DNN) method based on these game prediction approaches that trains a policy to play the game inside the learnt model. We learn a policy that, for many games, effectively performs the game in the real environment by means of numerous episodes of dataset (Q-table) aggregation, in which the policy is applied to gather more data (Q-table) in the original game.

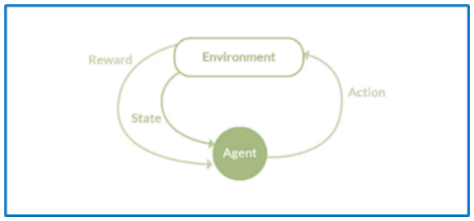


Figure 2 The most efficient way for an agent to learn how to interact with its surroundings in order to maximize the expected cumulative rewards of a task.

1.1. Related Work

Deep reinforcement learning (DRL) has evolved into a cutting-edge method to address various challenges in recent years. In their study, Li et al. (2024) [16] introduced a reinforcement learning-based routing algorithm specifically tailored for big-scale street networks, therefore proving the efficiency of DRL in managing dynamic and large-scale environments, Zi et al. (2024) [17]. Li M et al. (2023) [18] explored that DRL's uses go beyond the domain of transportation. Research by Li P et al. (2023) [19] in the field of data processing reveals that the advanced technologies of dual modal convolutional neural network and high-resolution semantic segmentation can successfully enable complicated data analysis, Li P (2018) [20].

Combining deep learning with reinforcement learning, Deep Reinforcement Learning (DRL) takes on difficult decision-making tasks. Robotics, transportation, video data analysis, and Natural Language Processing (NLP) are just a few of the areas where DRL has been effectively used (Wu K. et al. (2023)) [21]. Maximizing an objective within a rich environment of time requires determining the optimal policy (Liu, S. (2023)) [22]. To put DRL ideas into practice, many methods are utilized, such as Deep Deterministic Policy Gradient (DDPG), Deep Q Learning (DQN), and Markov Decision Processes (MDP). Performance of DRL-based implementations has recently been enhanced by Matsuo Y. et al. 2022 [23].

Many studies and discussions have focused on using machine learning (ML) for cloud scheduling, since traditional methods (non-ML) struggle to handle the complicated scheduling scenarios of cloud computing. For example, there has been work from Microsoft (Bianchini et al., 2020) [24], the CLOUDS Laboratory of The University of Melbourne (Ilager et al., 2021) [25], and other institutes (Duc et al., 2019; Rodrigues et al., 2020; Demirci, 2015) [26, 27, 28]. A new method that combines the best features of deep neural networks (DNNs) with reinforcement learning (RL), deep reinforcement learning (DRL) is a subfield of machine learning (ML). Recent research has shown that DRL is highly effective in handling Cloud scheduling problems and holds great advantages in numerous complex scenarios (Guo et al. 2021; Feng et al. 2020; Karthiban and Raj 2020; Raihen et al. 2024) [29, 30, 31, 32].

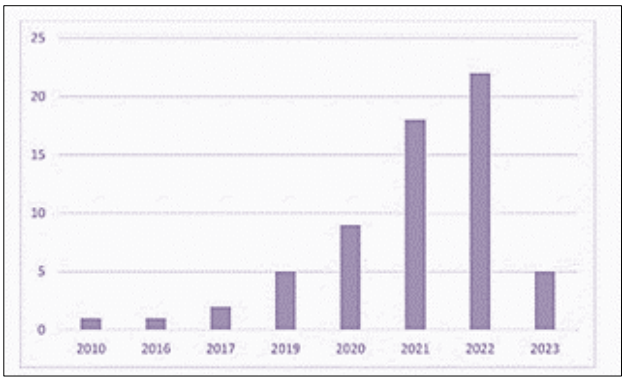


Figure 3 Increase in the number of reinforcement learning (RL) studies published. The data presented here are derived from Google Scholar searches and show the annual number of publications (y-axis) pertaining to RL (x-axis).

Achieving artificial general intelligence will be greatly aided by deep learning and reinforcement learning, which were named one of the 10 Breakthrough Technologies by the MIT Technology Review in 2013 and 2017, respectively. Leading AlphaGo developer David Silver even came up with a formula: AI = RL + DL (Silver, 2016) [33]. (Silver et al., 2016a; 2017) [34]. From 2019 to 2023, there was a noticeable increase in the usage of RL and DRL for maintenance planning tasks, as shown in Fig 3, which illustrates the number of publications per year based on RL and DRL for optimization and planning of maintenance over the previous thirteen (13) years. The number of articles discussing maintenance planning using RL and DRL has increased by more than 80%.

The remainder of this paper is structured in the following manner. In the following section, we will review the essential terminology and concepts that are associated with RL. This is followed by a discussion of the definition of interpretability (and explainability) within the broader context of artificial intelligence (AI) and machine learning, followed by an application of this definition within the framework of RL in Section 3. In the next parts, we will discuss the research that pertains to interpretable inputs and models, which are respectively presented in Subsection 3.1, 3.2, 3.3 and 3.4. This study's most important component, the work that addresses interpretable results and discussion, is discussed in Section 4, which is the most important element of this. For the Subsection 4.1, we also provide a review of deep Q-learning. In the Subsection 4.2, we provide instructions on how to train agents in environments that are complex, based on this overview. Finally, we conclude in Section 5.

2. Methodology

2.1. Reinforcement Learning (RL)

The implementation of reinforcement learning can be done in a variety of different methods. Both Q-Learning and Deep Q-Learning are excellent methods for accomplishing this goal. Q is an abbreviation for quality, which is a representation of the quality value of an action that our agent is able to perform.

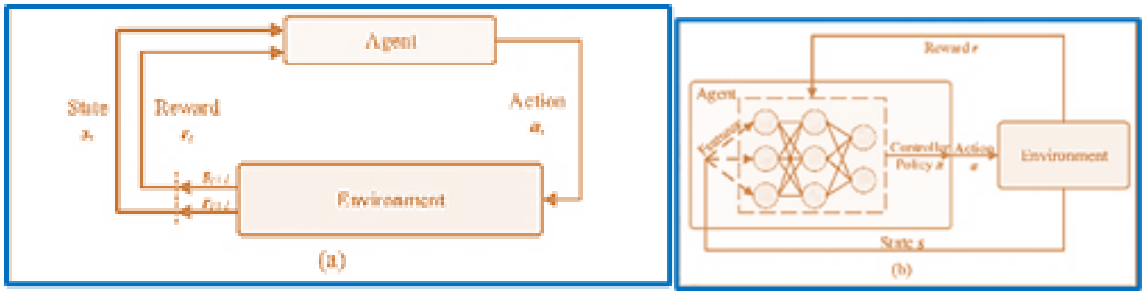


Figure 4 Training agents with deep neural networks (DNN) in reinforcement learning (RL)

Figure 4 shows that training an agent's policy to accomplish a task in an unfamiliar environment is the main objective of reinforcement learning. The environment provides the agent with observations and a reward, and the agent in turn transmits actions to the environment. The success of an action in relation to the task aim determines the reward. Most agents in the Reinforcement Learning Toolbox are built using deep neural networks, which are composed of several interconnected layers. These networks reflect default policies and value functions. In response to environmental action and observation specifications, the agents develop the network's inputs and outputs.

2.2. Markov Decision Process (MDP)

Markov decision processes (MDPs) provide a comprehensive framework for formalizing sequential decision-making problems. An MDP models a system where an agent interacts with an environment through a series of actions, leading to various states and associated rewards. In an MDP, the process unfolds over discrete time steps, where at each step, the agent observes the current state, selects an action based on a policy, transitions to a new state according to a probability distribution, and receives a reward. The goal is to find an optimal policy that maximizes the cumulative reward over time, considering both immediate and future rewards. This setup enables reinforcement of learning algorithms to learn and improve policies through exploration and exploitation of the environment. In the next subsections, we will provide a clear mathematical structure. MDPs are designed to allow the creation of efficient algorithms for the purpose of solving complicated decision-making problems in a variety of fields.

First, we discuss what a Markov process is. Markov states are those in which the transition to the next state, S_{t+1}, is dependent solely on the current state, S_t, and not on any of the states that came before it, $S_1\ S_2, \cdots, S_{t-1}$. So, for every t,

$$P[S_{t+1}|S_t] = P[s_{t+1}|S_1, S_2, \dots, S_t].$$

In RL, we talk about time-homogeneous Markov chains all the time. In these chains, the probability of the transition is copletely independent of t:

$$P[S_{t+1} = s'|S_t = s] = P[S_t = s'|S_{t-1} = s].$$

Now we can define formally,

Definition 1 (Markov Process)

A Markov Process is a tuple $(\mathcal{S}, \mathcal{P})$, where

$\mathcal{S}$ is a finite set of states.

$\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}_{ss'} = P[S_{t+1} = s' | S_t = s]$.

Here is how the Markov process's dynamics work: We begin in state S_0 and go to state S_1, which is chosen from the set $\mathcal{P}_{s_0 s_1}$. Next, we'll get to S_2 , which is derived from $\mathcal{P}_{s_1 s_2}$ and so on. This dynamic is illustrated by the following:

$$s_0 \longrightarrow s_1 \longrightarrow s_2 \longrightarrow s_3 \longrightarrow \cdots$$

A Markov decision process is the result of incorporating action, discount, and reward into a Markov process.

Definition 2 (Markov Decision Process)

A Markov decision process is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \sigma, \mathcal{R})$, where

$\mathcal{S}$ is a finite set of states.

$\mathcal{A}$ is a finite set of actions.

$\mathcal{P}$ is the state transition probability matrix, $\mathcal{P}^{a}_{ss'} = P[S_{t+1} = s' | S_t = s,\ A_t = a]$.

$\sigma \in [0,1]$ is called the discount factor.

$\mathcal{R}: \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function.

Within reinforcement learning, the MDP works as a model for the surrounding environment. If we take an action at state S_t in the MDP, the transition to state S_{t+1} will depend on both state S_t and the action you made at state $\mathcal{A}_t$. A reward function is also associated with every state-action pair.

This is how the MDP dynamic works: at the MDP, we begin at a state s_0 and select actions $a_0 \in \mathcal{A}$ to take. Our decision causes the MDP to randomly transition to a successor state s_1, chosen at random from the set $\mathcal{P}^{a_0}_{s_0 s_1}$. Another action, a_1, is then selected from state s_1. Here we are again at a state s_2, which is derived from $\mathcal{P}^{a_1}_{s_1 s_2}$. Next, we select a_2, and the process continues thereafter in the same manner. What follows is a representation of this sequential decision-making process:

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \cdots \longrightarrow$$

2.3. Policy, Value and Return Analysis

In RL, we want to select the best policy, or set of behaviors, over time in order to maximize the return's expected value. The terms "return" and "policy" are defined here. The sum of all discounts applied to the reward at time step t is known as the return G_t.

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

One way to look at discounted future benefits is as their current value. The immediate reward is valued above the delayed reward by the discount factor: A "myopic" evaluation is caused by $\boldsymbol{\gamma}$ being close to 0, while a "far-sighted" evaluation is caused by $\boldsymbol{\gamma}$ being close to 1.

A policy, denoted by the symbol π, is a distribution of acts among states,

$$\pi(a|s) = P[\mathcal{A}_t = a|\mathcal{S}_t = s].$$

Decisions at any one time are guided by policies.

Following this, we present the state-value function and the action-value function, which are two value functions. They are useful in determining the best course of action. From state s, the expected return is given by the state-value function v_π of an MDP, which is subsequently followed by policy π.

$$v_\pi(s) = E_\pi[G_t|S_t = s].$$

The value of state s in the long run when policy π is followed is provided by the state-value function $v_\pi(s)$. Both the immediate reward R_{t+1} and the discounted value of the successor state $\gamma v_\pi(S_{t+1})$ can be broken down into their respective components in the state-value function.

$$v_\pi(s) = E_\pi[R_{t+1}|S_t = s] + E_\pi[\gamma v_\pi(S_{t+1})|S_t = s].$$

Immediate

Discounted Value of Successor State

This is the action-value function the predicted outcome, denoted as $q_\pi(s, a)$, is derived from the initial state s, action a, and subsequent implementation of policy π.

$$q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a].$$

It is also possible to break down the action-value function in this way:

$$q_\pi(s, a) = E_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a].$$

To simplify, we define, $\mathcal{R}_s^a = E[R_{t+1}|S_t = s, A_t = a]$. Then we get a Bellman equation for v_π

$$v_\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s)\left(\mathcal{R}_s^a + \gamma \sum_{s'\in\mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')\right).$$

The bellman equation is utilized in Q-Learning, which can be expressed as the following notation that we used in our computation:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha\left(R_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)\right)$$

Learning Rate

Discount Factor

New State

Old State

Reward

With the help of this recursive function, we are able to modify the quality value of an action for each and every step that our agent will do.

2.4. Classic Q-learning Methods

Using a q-table is the way that classical q-learning is implemented. The brain of the agent is represented by this q-table. When it is first created, each row represents its own state, and each column represents a possible action that it can do while it is in that condition. The q-value of each action is taken into consideration by the agent, and it will choose the action that has the highest q-value among all of the actions chosen. In order to establish a reward for the action that was taken, the bellman equation will be utilized, and the q-table will be updated. The agent will continue to go through this process until they have completed their training. Following the completion of its training for **Frozen Lake Game**, we will be able to make use of the q-table that has been provided for any agent that is operating the environment that this q-table was created upon.

Table 1 In a Q-table, the horizontal axis of the table represents the actions, and the vertical axis represents the states. So, the dimensions of the table are the number of actions by the number of states.

	→	←	↑	↓
Start	0	0	0	0
Idle	0	0	0	0
Hole	0	0	0	0
End	0	0	0	0

2.4.1. Environment

To showcase the effectiveness of traditional Q-learning, the environment selected for this demonstration was the frozen lake game located next to the gymnasium. Traditional Q-Learning is particularly well-suited for environments that are predictable and uncomplicated, making it ideal for this test. The environment is designed as a static grid that consists of sixteen distinct states an agent can occupy, representing different positions on the lake.

The agent has four potential actions at its disposal: **moving up, down, left, or right**. Each action taken by the agent leads to a change in its state, depending on the direction chosen. The objective of the agent is to navigate across the grid from the starting point to a specified goal location without falling into any of the holes present on the grid. If the agent mistakenly steps into a hole, the episode immediately ends, and the agent incurs a penalty of losing a point. Conversely, if the agent successfully reaches the goal, the episode also ends, but in this case, the agent is rewarded with a point. This setup tests the agent's ability to learn and apply the optimal path to the goal while avoiding traps, demonstrating the practical utility of Q-learning in navigating environments with clear rules and fixed outcomes.

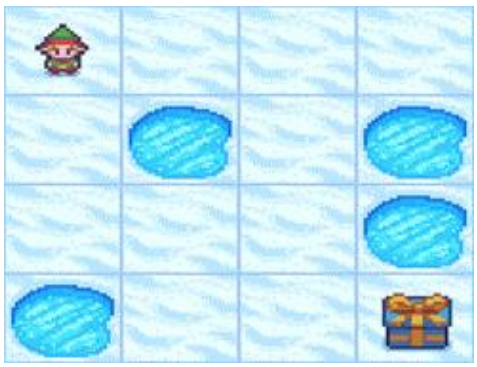


Figure 5 Evaluating traditional Q-learning in a static frozen lake game environment.

Fig 5. illustrates, in order to demonstrate the effectiveness of traditional q-learning, the environment that was selected was the **frozen lake environment** adjacent to the gymnasium. The traditional Q-Learning method thrives in contexts that are straightforward, such as this one. This static environment is made up of sixteen different states that the agent can be in, as well as four different actions that the agent can perform, which are up, down, left, and right respectively. It

is possible that the episode may come to an end and the agent will lose a point if they are unable to avoid falling into one of the holes. If the agent is successful in reaching the objective, then it will be awarded one point, and the episode will come to an end.

2.4.2. Implementations

To initiate the learning process, we establish a Q-table, effectively functioning as the cognitive center for our agent. At the outset, every entry within this Q-table is initialized to zero, signifying that the agent has no prior knowledge or experience of the environment. Structurally, the Q-table is designed with 16 rows, each corresponding to one of the different states the agent can encounter within the environment. Each state represents a unique location on the grid.

The table has rows and four columns. These columns are significant because they indicate the agent's four alternative actions—up, down, left, or right—based on its condition. This configuration lets the agent use the Q-table's developing values to determine the action that maximizes its reward in any state. These Q-values are updated when the agent interacts with the environment and earns rewards or penalties, directing it toward better goal-achieving tactics. Next, we initialize the discount rate (gamma) and learning rate, which are necessary for the Bellman equation. These factors are critical to Q-learning. Gamma, the discount rate, affects how much the agent prioritizes future benefits over immediate ones. How rapidly the agent updates its knowledge base depends on the learning rate. It controls how much fresh data replaces Q-table values. For this investigation, we set gamma as 0.6, suggesting moderate future reward valuation. The learning rate is 0.1, thus each new experience alters Q-values slightly, allowing for gradual learning.

After initializing the Q-table and learning parameters, we must build a method to balance exploration and exploitation. Effective learning requires this equilibrium. The agent explores by acting randomly. This method helps the agent find and record the results of different actions in different states without bias in the early stages of learning. **Exploration** gives the agent a comprehensive awareness of the world, revealing potential rewards in states it might have missed otherwise.

Exploitation uses the agent's Q-table experience to make decisions. The agent uses this knowledge to optimize its performance in familiar settings by choosing actions that maximize its rewards based on past results.

We employ epsilon, the exploration rate, to balance these two tactics. Training begins with epsilon set at 100%, meaning the agent investigates entirely and acts randomly. Random exploration lowers when the agent learns from its experiences and trusts the Q-table's recommendations.

We use a linear decay function on exploration rate to progressively switch the agent from exploration to exploitation. This function gradually decreases epsilon from 100% to 0% over episodes. As epsilon lowers, the agent is less likely to act randomly and more likely to use its learned experiences. This transition is adjusted so that by the time the exploration rate approaches zero, the agent has a strong awareness of the environment and can make the best Q-table decisions.
This systematic limitation in investigation provides a smooth transition from environmental information to maximizing rewards. This balanced approach is necessary for a successful, adaptable learning system. Formula for calculating epsilon:

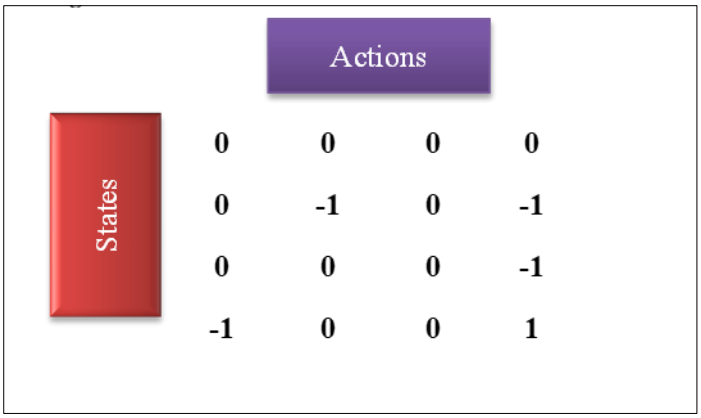
$$epsilon = minEpsilon + (MaxEpsilon - minEpsilon * Math.Exp(-epsilonDecay * episode)$$

Traditional Q-learning relies on a good incentive scheme to guide agent learning. Each strategy for implementing such systems is suited to the environment's needs and dynamics. A reward matrix is a frequent method. This matrix of rewards provides values for every possible environmental state. Each matrix item represents a state, and its reward value depends on its desirability. This systematization gives the agent constant feedback on its activities, helping it learn which are good and which are bad. Setting environmental limits that directly impact rewards is another method.

Environmental circumstances may dictate these limits. If the agent falls into a hole, a negative reward or penalty may be given. This instant input from the environment helps the agent quickly learn the implications of its actions, punishing bad choices and promoting good ones.

This study uses a reward matrix to take advantage of the simple surroundings. By setting clear incentives and penalties, we could match the agent's learning goals to the desired outcomes.

Table 2 By delivering clear, immediate, and appropriate feedback for each environmental activity, this reward matrix simplifies the agent's decision-making process and improves learning efficiency. This organized method helps the agent quickly grasp and adapt to its surroundings, maximizing performance through intelligent decision-making.



In addition, The **Bellman equation** is utilized to derive the Q-table, and the following algorithm illustrates its application. The best policy and value function are represented by MDP. The maximum value function across all policies is the best state-value function $v_*(s)$, i.e., $v_*(s) = \underbrace{max}_{\pi} v_\pi(s)$.

Python code for computing a q-table

```
var maxQ = double.MinValue;
//get q max of next state poss values
for (int i = 0; i < possNextNextStates.Count; i++)
{
    var possStates = possNextNextStates[i];
    var q = QTable[qNextState, possStates.Item3];
    if (q > maxQ) { maxQ = q; }
}

QTable[qState, action.Item3]
    = QTable[qState, action.Item3] * (1 - learnRate) + learnRate * (reward[action.Item1, action.Item2] + gamma * maxQ);
```

The max Q-value in Q-learning represents future rewards. This value is crucial because it shows the agent's maximum future reward given its current condition and decisions. We find max-Q by selecting the agent's subsequent state's highest Q value from all conceivable actions. This strategy assumes the agent will optimize its long-term outcomes by following the path of maximum rewards.

To evaluate and include future benefits into current decision-making, multiply max Q-value by discount factor gamma. This discount factor, which we initialized, measures how much future rewards are valued above immediate ones. A greater gamma indicates that the agent values future rewards, favoring long-term profits above short-term gains. This technique relies on the Bellman equation. It integrates these calculated future benefits (discounted maxQ) with immediate rewards from specific activities to update the Q-table Q-values. It does so to ensure that Q-values reflect both direct and future advantages of actions, allowing the agent to make better informed decisions that optimize long-term results. Q-learning algorithms rely on the Bellman equation's dynamic Q-table update for learning and decision-making.

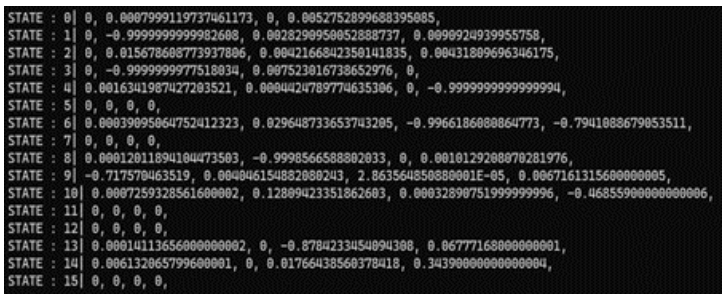
3. Result and Discussion

This training regimen lets us examine the Q-table and track our agent's training. This study is significant because it shows the agent's learning and adaptability to the environment through its decision-making. Our environment is interesting since it has two paths to the objective, each with six stages. A dual-path structure tests the agent's decision-making techniques under choice and equivalency conditions. Since both courses are equally viable in terms of steps, the agent's choice reveals its preference patterns and learning efficiency.

By studying how the agent chooses between these paths, we can understand the effectiveness of the Q-learning algorithm in balancing exploration (testing new paths) and exploitation (using known paths). This also helps in assessing how well the agent optimizes its path based on past experiences stored in the Q-table, and how these decisions evolve as the agent progresses through more episodes of learning. The ability to choose allows the agent not only to find

the shortest route to the goal but also to explore alternative strategies that might yield better long-term learning outcomes.

3.1. The outcome of utilizing the optimal strategy



This outcome illustrates the training environment utilized in the Q-learning algorithm, showcasing a grid with two distinct paths leading to the goal, each comprising six steps. The grid is clearly labeled to indicate the starting point, the two available routes, and the goal. The figure also includes a visual representation of the Q-table at various stages of the learning process, highlighting how the agent's decisions and path preferences evolve over time. Annotations on the paths indicate the choices made by the agent, providing insight into its strategy of exploration versus exploitation. This visual representation aids in understanding the dynamics of agent learning and decision-making within a structured environment.

Theorem 1 [Han, X. 2018] [35]: For any Markov Decision Process,

There exists an optimal policy π_* that is better than or equal to all the other policies, $\pi_* \geq \pi$, for all π.

All optimal policies achieve the optimal value function, $v_{\pi_*}(s) = v_*(s)$.

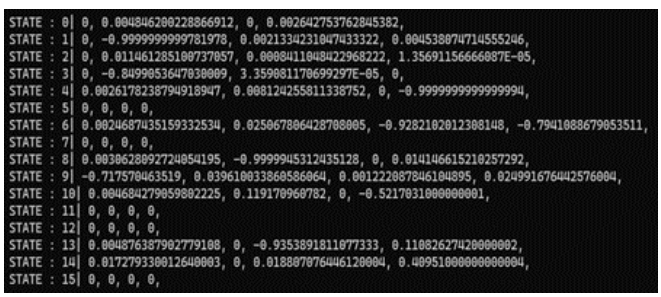
All optimal policies achieve the optimal action-value function, $q_{\pi_*}(s,a) = q_*(s,a)$.

From the **Theorem 1**, we can find optimal policy by maximizing $q_*(s,a)$ ovel all actions,

$$\pi_*(a|s) = \begin{cases} 1 & if\ a = \underbrace{arg\ max}_{a \in \mathcal{A}}\ q_*(s,a) \\ 0 & otherwise \end{cases}$$

In this case, our agent decided to start from the very beginning. The Q-values for each possible action are displayed in the figure, separated by commas, and correspond to the following actions in order: up, down, left, and right. By examining state 0, which is the agent's initial position, we can see the Q-values associated with each action. Specifically, the Q-value for moving right is 0.005, while the Q-value for moving down is 0.0007.

These Q-values represent the learned quality or utility of taking each action from that state, with higher values indicating more favorable outcomes based on the agent's experience. Given these values, the agent will select the action with the highest Q-value. In this case, the agent will choose to move right, as it has the highest Q-value of 0.005, indicating it is the optimal action to take according to the current knowledge stored in the Q-table. This decision-making process exemplifies how the agent leverages its learned Q-values to navigate the environment effectively. We can take another instance where the agent chooses to go down instead of right,



In this case, the quality of the down action is 0.004, which is higher than the quality of the right action, which is **0.002**. This highlights that the Q-values, representing the estimated value of each action, can differ significantly depending on the outcomes of the training process. Each training session may result in different experiences and rewards for the agent, leading to variability in the Q-values. This variability illustrates how the agent's learning process adapts to the specific conditions and paths encountered during each session, ensuring that the agent continuously improves and refines its strategy based on diverse experiences. Thus, the Q-values obtained at the conclusion of each training instance will invariably be distinct from one another, reflecting the dynamic nature of the learning process.

3.2. Deep Q-learning

Traditional Q-learning is effective for simple environments, but it becomes impractical for more complex scenarios due to the limitations of the Q-table. In traditional Q-learning, a Q-table is used to store the Q-values, which represent the expected utility of taking a particular action in a particular state. However, as the complexity of the environment increases, the number of possible states and actions grows exponentially. This results in a Q-table that is excessively large, making it highly inefficient for the agent to search and update constantly.

In a complex environment, the agent would need to store and retrieve Q-values for an immense number of state-action pairs, which becomes computationally infeasible. This is where Deep Q-learning comes into play. Deep Q-learning replaces the Q-table with a neural network, which acts as the agent's decision-making "brain." The neural network takes the current state as input and outputs Q-values for each possible action, allowing the agent to determine the optimal action to take.

One of the key advantages of Deep Q-learning is its ability to generalize across similar states. While a Q-table requires explicit storage and updates for each state-action pair, a neural network can learn patterns and relationships within the data, enabling it to make predictions about unseen states based on its training. This significantly reduces the amount of memory and computation required, making it feasible to handle much more complex environments.

In traditional Q-learning, the Q-values for every action are updated at each step based on the agent's experience. In contrast, Deep Q-learning updates the weights of the neural network. The agent uses a technique called experience replay, where it stores its experiences (state, action, reward, next state) in a replay buffer. The neural network is then trained by sampling random batches from this buffer, which helps stabilize and improve the learning process.

Additionally, Deep Q-learning often employs techniques like target networks and double Q-learning to further enhance stability and performance. The target network, a separate copy of the neural network, is used to compute target Q-values, which are more stable and help reduce oscillations during training. Double Q-learning helps to mitigate the overestimation bias present in standard Q-learning by decoupling action selection from action evaluation.

These weights determine the influence each node in a neural network has on the output. The neural network takes the current state as input and produces Q-values for all possible actions. These Q-values represent the expected utility of each action in that state. The agent then selects the action with the highest Q-value, indicating the most advantageous choice given the current circumstances.

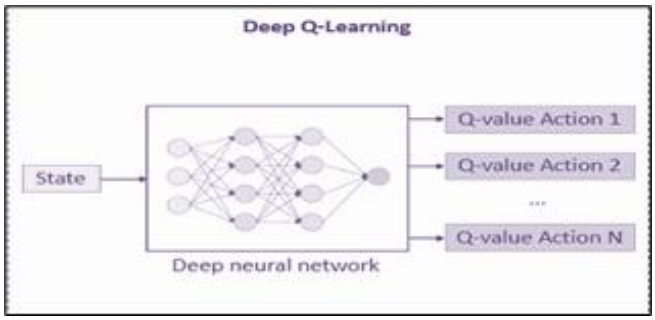


Figure 6 A deep neural network will estimate the Q-values for each state-action combination in a given environment, and the network will then approximate the optimal Q-function.

3.2.1. Environment

This environment was developed using the Unity game engine. It features our agent, which is represented by a blue capsule, three goals indicated by green circles, and an enemy depicted as a red triangle. The agent navigates the

environment, aiming to reach the goals while avoiding the enemy. The AI for both the goals and the enemy is intentionally straightforward. The green goals move randomly around the map, and when our agent reaches a certain proximity, the goals move away to maintain a distance. On the other hand, the red triangle, representing the enemy, roams the map freely but becomes aggressive and starts chasing our agent as soon as it gets within a specific range.

This environment is designed to help our agent learn critical skills. It must navigate the map, avoid the pursuing red triangle, and reach the three green goals as efficiently as possible. Through this setup, the agent learns to balance the dual objectives of evading the enemy and achieving the goals in the shortest amount of time, thereby developing effective strategies for navigation and threat avoidance.

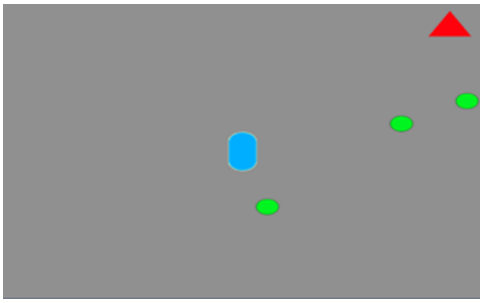


Figure 7 Training an agent to navigate goals and avoid enemies in a dynamic environment.

In Figure 7, in the described environment, a blue capsule represents our agent, green circles represent three goals, and a red triangle represents an enemy. The green goals move randomly but flee when the agent gets close. The red triangle roams the map and chases the agent when it comes within range. The setup aims to train the agent to avoid the enemy while reaching the goals quickly.

3.2.2. Rewards

The most crucial component of reinforcement learning is the implementation of rewards. If the rewards are not appropriately calibrated, the training process can become highly inefficient and take significantly longer to converge. Furthermore, Q-Learning models are particularly adept at exploiting the environment to maximize rewards, which can sometimes result in unintended or undesirable behaviors from the agent. To address these challenges and guide the agent towards the desired outcomes, the following incentives have been established in this setting:

Figure 8 Designing Rewards and Punishments in the Environment for Corresponding Actions**.**

The rewards were designed to maintain a consistent balance between rewards and punishments, which is crucial for optimizing the training process. By carefully calibrating the rewards, we can ensure that the agent learns efficiently and avoids undesirable behaviors.

To motivate the agent to collect goals quickly, we impose a slight penalty for each second it remains active. This penalty is set at -0.1 points per frame. Given that the simulation runs at 60 frames per second, the agent loses 6 points per second. This continuous penalty encourages the agent to complete its objectives in the shortest time possible, promoting more efficient learning and faster goal collection. By structuring the rewards and penalties in this manner, we aim to create a more effective training environment that fosters desirable behaviors and minimizes inefficient actions.

3.2.3. Training Process

To accelerate the training process, we run multiple environments simultaneously. This setup significantly enhances the speed at which our agent learns by allowing it to gather more experience in a shorter amount of time.

Figure 9 Visualizing agent training progress: from frequent failures to consistent success.

In Fig 9., the left image shows the agent starting training. Agents caught by the red enemy in black squares fail to achieve their goals. A gray square indicates that the agent achieved all three goals in that setting. Right image shows the agent after training. Black squares have decreased significantly, with most squares becoming gray. This graphic transition shows that the agent can maneuver, avoid the red adversary, and reliably gather all three goals.

By training across multiple environments simultaneously, the agent can experience a diverse range of scenarios and challenges. This approach not only speeds up the learning process but also ensures a more robust and generalized skill set for the agent, ultimately leading to better performance in achieving its objectives.

The graph provided illustrates the learning pattern of our agent throughout its training process by depicting the duration of each episode. In this context, an episode concludes when the agent either successfully collects all three goals or gets caught by the red triangle. The graph features two distinct training models, allowing us to compare their performance.

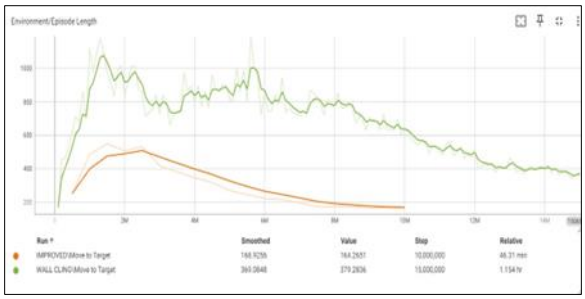


Figure 10 Graphing the agent's learning curve: episode duration and performance evolution.

As **Fig 10.** demonstrates the provided graph illustrates the learning pattern of our agent throughout its training process by depicting the duration of each episode. This visualization helps us understand how the agent's performance evolves over time, highlighting key phases of learning, including initial struggles, gradual improvements, and eventual efficiency gains. It offers insights into the agent's ability to adapt and optimize its actions within the environment, providing a clear picture of the training effectiveness and the impact of different learning strategies.

We can focus on the orange line first. At the beginning of the training, the episode lengths are very short because the agent is frequently caught by the red triangle. This indicates that the agent has not yet learned effective strategies for avoiding the enemy. However, as training progresses, we observe an increase in episode length. This upward trend signifies that the agent is starting to learn how to evade the red triangle, surviving longer within each episode. Eventually, as the training continues, the agent begins to efficiently collect the goals. This newfound efficiency is reflected in the decreasing episode lengths, as the agent becomes quicker at achieving all three goals. The decline in episode length demonstrates that the agent is not only avoiding the enemy more effectively but also optimizing its path to collect the goals faster, hence completing episodes more swiftly.

In contrast, the green line represents a model where the reward system was not properly configured. While it does show an initial peak, indicating that the agent is beginning to learn to avoid the red triangle, the pattern is inconsistent

compared to the orange line. The fluctuations in the green line suggest that the agent's learning process is less stable and effective. It fails to consistently avoid the enemy and collect the goals as efficiently as the model represented by the orange line.

This comparison underscores the critical importance of correctly setting and balancing rewards in reinforcement learning. A well-calibrated reward system, as shown by the performance of the orange line, leads to more efficient and effective learning. The agent can consistently improve its strategies, resulting in longer episodes initially (as it learns to survive) and shorter episodes later on (as it learns to complete its objectives quickly). Conversely, the poor performance of the green line model highlights how inadequate reward handling can lead to suboptimal learning, with the agent failing to develop consistent and efficient behaviors. By examining these patterns, we can better understand the impact of reward structures on the training process and the overall performance of reinforcement learning agents.

4. Conclusion

In conclusion, this study highlights the distinct mechanisms and applications of traditional Q-learning and Deep Q-learning. Both methods are grounded in the principles of reinforcement learning, where agents learn to make decisions by receiving and maximizing rewards from their interactions with the environment.

Q-learning employs the Bellman equation to iteratively update the Q-values, which represent the expected utility of taking a specific action in a given state. Traditional Q-learning uses a Q-table to store these Q-values. While this approach is effective for simple environments, it struggles to scale with increasing complexity due to the exponential growth of possible state-action pairs. The Q-table becomes impractically large, making it inefficient for complex environments.

Deep Q-learning addresses these scalability issues by leveraging neural networks to approximate the Q-values. Instead of maintaining a vast Q-table, the neural network takes the current state as input and outputs Q-values for all possible actions. The agent then selects the action with the highest Q-value. This approach allows Deep Q-learning to handle complex environments with a high dimensionality of states and actions more efficiently. The neural network in Deep Q-learning effectively replaces the Q-table, acting as the brain of the agent. It learns to predict the Q-values through training, adjusting its weights based on the rewards received from the environment. This enables the agent to develop sophisticated strategies for navigating the environment and achieving its goals.

Reinforcement learning, whether through traditional Q-learning or Deep Q-learning, provides valuable insights into how agents can learn and adapt to their surroundings. However, due to their optimization capabilities, agents can sometimes exploit the environment in ways that lead to unintended or undesirable behaviors. This phenomenon underscores the importance of carefully designing reward structures and considering potential edge cases during the development of reinforcement learning systems.

By understanding the strengths and limitations of both traditional Q-learning and Deep Q-learning, we can better apply these techniques to various problems, ensuring efficient learning and desirable outcomes in both simple and complex environments.

Compliance with ethical standards

Disclosure of conflict of interest

No conflict of interest to be disclosed.

References

[1] Y. Li, “Deep reinforcement learning: An overview”, arXiv preprint arXiv:1701.07274.

[2] Reinforcement Learning, online, Wikipedia, Reinforcement learning - Wikipedia.

[3] M. N. Raihen, and S. Akter, “Prediction modeling using deep learning for the classification of grape-type dried fruits” International Journal of Mathematics and Computer in Engineering, vol.2, no. 1, 2024, 1-12.

[4] M. N. Raihen, S. Akter, F. Tabassum, F. Jahan, and S. Begum, A statistical analysis of excess mortality mean at Covid-19 in 2020-2021”, Computational Journal of Mathematical and Statistical Sciences, vol. 2, no. 2, 2023, 223-239.

[5] M. N. Raihen, and S. Akter, "Comparative Assessment of Several Effective Machine Learning Classification Methods for Maternal Health Risk" Computational Journal of Mathematical and Statistical Sciences, vol. 3, no.1, 2024, 161-176.

[6] M. N. Raihen, and S. Akter, "Forecasting Breast Cancer: A Study of Classifying Patients' Post-Surgical Survival Rates with Breast Cancer" Journal of Mathematics and Statistics Studies, vol. 4, no. 2, 2023, 70-78.

[7] S. Whiteson, B. Tanner, M. E. Taylor, and p. Stone, "Protecting against evaluation overfitting in empirical reinforcement learning" In 2011 IEEE symposium on adaptive dynamic programming and reinforcement learning (ADPRL), 2011, (pp. 120-127). IEEE.

[8] M. c. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. Hausknecht, and M. Bowling, "Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents" Journal of Artificial Intelligence Research, vol. 61, 2018, 523-562.

[9] M. N. Raihen, S. Akter, F. Tabassum, F. Jahan, and M. N. Sardar, "A statistical analysis of positive excess mortality at Covid-19 in 2020-2021" Journal of Mathematics and Statistics Studies, vol. 4, no. 3, 2023, 07-17.

[10] A. L. Boulesteix, S. Lauer, and M. J. Eugster, "A plea for neutral comparison studies in computational sciences" PloS one, vol. 8, no. 4, 2013, e61562.

[11] V. Stodden, F. Leisch, and R. D. (Eds) Peng, "Implementingreproducible research", Vol. 546, 2014, Boca Raton, FL: Crc Press.

[12] R. R. Bouckaert, and E. Frank, "Evaluating the replicability of significance tests for comparing learning algorithms", In Pacific-Asia conference on knowledge discovery and data mining, 2004, pp. 3-12 Berlin, Heidelberg: Springer Berlin Heidelberg.

[13] R. R. Bouckaert, "Estimating replicability of classifier learning experiments", In Proceedings of the twenty-first international conference on Machine learning, 2004, p. 15.

[14] R. Vaughan, and J. Wawerla, "Publishing identifiable experiment code and configuration is important, good and easy", 2012, *arXiv preprint arXiv:1204.2235*.

[15] M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey" Journal of Artificial Intelligence Research, Vol. *4*, no. 1, 1996, pp. 237-285.

[16] D. Li, Z. Zhang, B. Alizadeh, Z. Zhang, N. Duffield, M. A. Meyer, ... and A. H. Behzadan, "A reinforcement learning-based routing algorithm for large street networks", International Journal of Geographical Information Science, vol. *38, no.* 2, 2024, pp. 183-215.

[17] Y. Zi, Q. Wang, Z. Gao, X. Cheng, and T. Mei, "Research on the application of deep learning in medical image segmentation and 3d reconstruction", *Academic Journal of Science and Technology*, vol. *10*, no. 2, 2024, pp. 8-12.

[18] M. Li, B. Belzile, A. Imran, L. Birglen, G. Beltrame, and D. St-Onge, "From assistive devices to manufacturing cobot swarms", In 2023 32nd IEEE International Conference on Robot and Human Interactive Communication (RO-MAN), 2023, pp. 234-240. IEEE.

[19] P. Li, M. Abouelenien, and R. Mihalcea, "Deception Detection from Linguistic and Physiological Data Streams Using Bimodal Convolutional Neural Networks", 2023, *arXiv preprint arXiv:2311.10944*.

[20] P. Li, Y. Lin, and E. Schultz-Fellenz, "Contextual hourglass network for semantic segmentation of high resolution aerial imagery", 2018, *arXiv preprint arXiv:1810.12813*.

[21] K. Wu, "Creating panoramic images using ORB feature detection and RANSAC-based image alignment", *Advances in Computer and Communication*, vol. *4*, no. 4, 2023, pp. 220-224.

[22] S. Liu, K. Wu, C. Jiang, B. Huang, and D. Ma, "Financial time-series forecasting: Towards synergizing performance and interpretability within a hybrid machine learning approach", 2023, *arXiv preprint arXiv:2401.00534*.

[23] Y. Matsuo, Y. LeCun, M. Sahani, D. Precup, D. Silver, M. Sugiyama, ... and J. Morimoto, "Deep learning, reinforcement learning, and world models", Neural Networks, Vol. *152*, 2022, pp. 267-275.

[24] R. Bianchini, M. Fontoura, E. Cortez, A. Bonde, A. Muzio, A. M. Constantin, ... and M. Russinovich, "Toward ml-centric cloud platforms", Communications of the ACM, vol. *63*, no. 2, 2020, pp. 50-59.

[25] S. Ilager, K. Ramamohanarao, and R. Buyya, "Thermal prediction for efficient energy management of clouds using machine learning", IEEE Transactions on Parallel and Distributed Systems, vol. *32*, no. 5, 2020, pp. 1044-1056.

[26] T. L. Duc, R. G. Leiva, P. Casari, and P. O. Östberg, "Machine learning methods for reliable resource provisioning in edge-cloud computing: A survey", ACM Computing Surveys (CSUR), vol. *52*, no. 5, 2019, pp. 1-39.

[27] T. K. Rodrigues, K. Suto, H. Nishiyama, J. Liu, and N. Kato, "Machine learning meets computation and communication control in evolving edge and cloud: Challenges and future perspective", IEEE Communications Surveys & Tutorials, vol. *22, no.* 1, 2020, pp. 38-67.

[28] M. Demirci, "A survey of machine learning applications for energy-efficient resource management in cloud computing environments", In 2015 IEEE 14th international conference on machine learning and applications (ICMLA), 2015, pp. 1185-1190, IEEE.

[29] W. Guo, W. Tian, Y. Ye, L. Xu, and K. Wu, "Cloud resource scheduling with deep reinforcement learning and imitation learning", IEEE Internet of Things Journal, vol. *8*, no. 5, 2021, pp. 3576-3586.

[30] J. Feng, F. R. Yu, Q. Pei, X. Chu, J. Du, and L. Zhu, "Cooperative computation offloading and resource allocation for blockchain-enabled mobile-edge computing: A deep reinforcement learning approach", IEEE Internet of Things Journal, vol. 7, no. 7, 2019, pp. 6214-6228.

[31] K. Karthiban, & J. S. Raj, "An efficient green computing fair resource allocation in cloud computing using modified deep reinforcement learning algorithm", Soft Computing, vol. 24, no. 19, 2020, pp. 14933-14942.

[32] Raihen, M.N. and Akter, S., "Sentiment analysis of passenger feedback on US airlines using machine learning classification methods'', vol. 23, no. 1, 2024, pp.2260-2273.

[33] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, ... and d. Hassabis, "Mastering the game of Go with deep neural networks and tree search", nature, vol. 529, no. 7587, 2016, pp. 484-489.

[34] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, ... and D. Hassabis, "Mastering the game of go without human knowledge", nature, vol. 550, no. 7676, 2017, pp. 354-359.

[35] X. Han, "A mathematical introduction to reinforcement learning", Semantic Scholar, 2018, pp. 1-4.