The Hybrid Multi-Task Deep Reinforcement Learning System Optimized from Distinct Perspectives

Hsu Zarni Maung

Faculty of Computer Science

University of Computer Studies, Mandalay

2023

The Hybrid Multi-Task Deep Reinforcement Learning System Optimized from Distinct Perspectives

Hsu Zarni Maung

Bachelor of Computer Science, UCS(Kalay) 2020

Faculty of Computer Science

University of Computer Studies, Mandalay

2023

The Hybrid Multi-Task Deep Reinforcement Learning System Optimized from Distinct Perspectives

Hsu Zarni Maung

Faculty of Computer Science

University of Computer Studies, Mandalay

A dissertation submitted to the University of Computer Studies, Mandalay

in partial fulfillment of the requirements for the degree of

Master of Computer Science

June 2023

Programme Authorized to Offer Degree:

Computer Science

Approval

Name:	**Hsu Zarni Maung**
Degree:	**Master of Computer Science**
Title:	**The Hybrid Multi-Task Deep Reinforcement Learning System Optimized from Distinct Perspectives**
Examining Committee:	**Chair: Dr. Khin Thanda Soe** Acting Rector
	Dr. Yi Yi Hlaing Board of Examiner Member Professor and Head Faculty of Computer Science
	Dr. Win Thuzar Kyaw Board of Examiner Member Lecturer Faculty of Computer Science
	Dr. Myo Khaing Supervisor Professor Faculty of Computer Science Head of the Department of Career
	Dr. Thandar Htwe External Examiner Professor and Head Data Center Myanmar Institute of Information Technology
Date Defended/Approved:	June 27, 2023

Statement of Originality

The work contained in this dissertation has not been previously submitted for a degree or diploma at any other university or educational institution.

Date **Hsu Zarni Maung**

Abstract

Artificial Intelligence (AI) goes further still: the evaluation of Reinforcement Learning (RL) has been elevated to a conceptually simple and lightweight framework with deep learning. This new direction is named Deep Reinforcement Learning (DRL), which is famous for achieving high performance in a variety of environments and tackling challenging, complex, and demanding tasks. Focusing on single task learning is quite inefficient in data-rich environment although RL intelligent agents designed with model-free-based approaches achieved performance optimization. The limitations of applications of DRL algorithms are hungry a lot of data, a lot of computations and a lot of training time. To mitigate these issues is by applying multi-task learning-oriented approach for the optimization of DRL agents. The objective of this thesis is to present asynchronous standard reinforcement learning algorithms to many scenarios across related tasks from same distribution on OpenAI Gym based Atari 2600 video game environment. The Hybrid Multi-Task Asynchronous Advantage Actor-critic (A3C) is accomplished by making many parallel copies of the agent and executing them asynchronously in different threads, allowing each copy to independently explore and learn from the environment. The actor network and the critic network are the two neural networks that the algorithm uses. Based on the current state of the environment, the actor network is in charge of choosing what to do. By calculating the expected return from states, the critic network assesses the value of action. The main goal is to show that parallel actor-learners have a stabilizing effect on training on different video games and to build a hybrid multi-task learning model operating within differences in semantically similar environments with related work. Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) are used in implementing Actor and Critic networks for each worker and the global network. A Long Short-Term Memory layer is inserted after the final fully linked layer in Hybrid A3C models. This significantly improved rewards in some environments, such as Breakout, Pong, and River Raid. Although two environments were being trained with each global step, the average time is the same with single-environment A3C.

Keywords: Machine learning; deep reinforcement learning; neural networks; transfer learning; actor mimic; multi-agent systems.

Acknowledgements

First of all I would like to thank to my parents who mainly supported me to complete this thesis and thank for their kindness and encouragements. I am also graceful to all respectable people who directly or indirectly contributed towards the success of this thesis.

I would like to thank and pay respect to **Dr. Khin Thanda Soe**, Acting Rector of the University of Computer Studies, Mandalay, for allowing me to develop this thesis and giving me general guidance during the period of study.

My thanks and respects go to **Dr. Win Thuzar Kyaw**, Lecturer, Faculty of Computer Science, University of Computer Studies, Mandalay, for her valuable guidance and administrative support, as Dean of Master Course, throughout the development of the thesis.

I am extremely grateful to my supervisor, **Dr. Myo Khaing**, Professor, Faculty of Computer Science, Head of the Department of Career, University of Computer Studies, Mandalay, for his valuable recommendations, his detailed guidance, his patient supervision and support throughout the preparation of the thesis.

I also would like to express my special thanks to **Daw Nilar Aung,** Lecturer, Department of Languages, University of Computer Studies, Mandalay, for editing this thesis from the language point of view.

Finally, thanks are also extended especially to all my university teachers, childhood teachers, seniors, friends, juniors and staffs for their encouragement and cooperative help for the completion of the thesis dissertation.

Table of Contents

List of Tables

List of Figures

List of Acronyms

AI	Artificial Intelligence
AGI	Artificial General Intelligence
ML	Machine Learning
DL	Deep Learning
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
MTDRL	Multi-Task Deep Reinforcement Learning
TL	Transfer Learning
A3C	Asynchronous Advantage Actor-Critic
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
RMSprop	Root Mean Squared Propagation
NLP	Natural Language Processing
ALE	Arcade Learning Environment
FC	Fully-Connected
ANN	Artificial Neural Network
IMPALA	Importance Weighted Actor-Learner Architecture
DQN	Deep Q Learning
TD	Temporal Difference
TDL	Temporal Difference Learning
MLP	Multi-Level Perception

Chapter 1

Introduction

Modern Artificial Intelligence (AI) approach, which is “the ability to learn or understand or to deal with new or trying situation”, goes on progressing various fields. Deep Reinforcement Learning (DRL) is more capable of learning raw data as input from the environment, which opens up more applications as shown in Figure 1.1, such as games, healthcare, education, transportation, business management, robotics, computer vision, Natural Language Processing (NLP), science engineering art, etc. The two characteristics of Reinforcement Learning (RL) are trail-and-error search and delay reward. Every RL agent can sense aspect of their own environments, has explicit goals in order to choose actions to influence their environments.

With parallel reinforcement learning, a set of closely related tasks from the operating environment will be learned simultaneously by individual agents with Asynchronous Advantage Actor-Critic (A3C). A3C consists of multiple independent agents with their own networks and weights who interact with different copies of the environment in parallel. The global network shares its parameters with all of the individual networks of agents after deriving new parameters by combining the learning parameters of all the individual agents. After each had updated, the agents reset their parameters to those of the global network and continue their independent exploration and training for *n* steps until they update themselves again.

This thesis highlights the integration of multi-task learning and deep reinforcement learning within the A3C framework, leading to a hybrid algorithm that leverages distinct perspectives for optimization. Specifically, the Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) layers are employed to implement the Actor and Critic networks, respectively, in the hybrid multi-task A3C system. This combination enables the agent to capture both spatial and temporal dependencies, enhancing its understanding of complex environments and improving decision-making capabilities. The motivation behind hybrid multitask A3C with LSTM is to enhance the learning capabilities of reinforcement learning agents by enabling them to simultaneously learn and perform multiple related tasks.

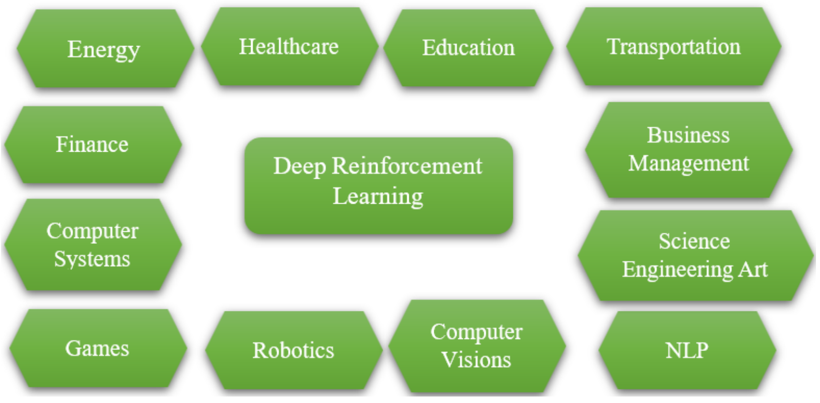


Figure 1.1. Application of Deep Reinforcement Learning

1.1. Optimization of Deep Reinforcement Learning

DRL has the disadvantage of only being applicable to certain professions; as a result, if it were to scale to other contexts, it would be difficult. RL has a frustrating propensity to over-fit objectives, and the intelligence will readily slip through the cracks and result in unexpected outcomes. The reduction from learning to optimization is less straightforward in RL than it is in supervised learning. One difficulty is that full analytic access is needed to the function trying to be optimized; the agent expects total reward; this objective also depends on the unknown dynamics model and reward function. Another difficulty is that the input data of the agent strongly depends on its behavior, which makes it hard to develop algorithms with monotonic improvement. This is a typical reinforcement learning issue known as the exploration-exploitation dilemma. Data comes from the present approach, but if the agent investigates it too much, he or she (agent) will end up with a ton of meaningless data and no means to glean anything meaningful from it. Additionally, if the agent exerts too much effort, the agent will not find the best move.

One of the most well-known standards for deep reinforcement learning is found in Atari games. Researchers can play various Atari games with human or superhuman performance using some optimization approaches. Video games are the ideal environments for AI research since different games present engaging and challenging tasks for agents to solve. Not only simple experimentation, adaptable development, portability and dependability, and reproducibility, but also training a virtual agent to do better than human competitors can tell how to improve various procedures in a variety of fascinating subfields. The agents can make mistakes in controlled conditions, but they are unable to make mistakes and learn from them in actual situations.

1.2. Statement of the Problem

The best model for RL agents is an improvement and implementation of the state-of-the-art hybrid multi-task Learning Model (A3C), which has several distinct differences from Asynchronous one-step Q-learning, Asynchronous one-step Sarsa, and asynchronous n-step Q-learning [1].

Traditional DRL approaches struggle with training multiple tasks in parallel as they tend to interfere with each other during the learning process. This interference often results in suboptimal policies and reduces overall performance. The tasks included in a multi-task learning setup can vary significantly in complexity, reward structures, and optimal strategies. Designing a system that can efficiently handle task heterogeneity which is essential to achieve strong performance across all tasks. As the number of tasks increases, the scalability of the multi-task DRL system becomes crucial. The proposed solution should be able to accommodate a growing number of tasks without sacrificing training efficiency and performance. Balancing exploration and exploitation are critical aspects of DRL. In multi-task learning, this trade-off becomes even more intricate as the agent needs to explore the state space of multiple tasks simultaneously. An effective multi-task DRL system should be able to leverage knowledge gained from learning one task to improve performance on other related tasks, fostering transfer learning between tasks.

The first challenge is to adapt effectively tasking heterogeneity, ensuring that all tasks receive appropriate attention and learning resources based on their complexities. Agent faces the challenge of scaling efficiently with an increasing number of tasks without compromising the quality of the learned policies. Agents also need to foster transfer learning between tasks to enhance their performance on related tasks using knowledge gained from other tasks.

1.3. Motivation and Objectives

Deep Reinforcement Learning has achieved remarkable success in controlling complex tasks, as demonstrated by groundbreaking achievements in games like AlphaGo, MuZero, and AlphaZero, where they surpassed professional human players [2]. The game of Go serves as a prime example of the challenges that artificial intelligence must overcome. It involves difficult decision-making, an unsolvable search problem, and an optimal solution that is incredibly intricate, making it seemingly impossible to directly approximate using a policy or value function [3]. The LSTM component in hybrid multitask A3C allows the agent to capture and retain information over longer sequences, which can be crucial for tasks that involve temporal dynamics or dependencies. It enables the agent to remember relevant past experiences and make more informed decisions based on the current context.

In real environment, the agent is completely unfamiliar at given time because of noisy and inaccurate sensors or missing parts of the state from the sensor data. The objective of hybrid multitask A3C with LSTM is to improve the agent generalization and transfer learning abilities. By training the agent on multiple tasks simultaneously, it can learn to extract common features and knowledge that are beneficial across tasks.

Firstly, it focuses on exploring and exploiting the problem space more comprehensively, particularly in scenarios where partial observation is involved. By effectively utilizing available information and making informed decisions, the system aims to enhance its understanding and performance in complex environments. Secondly, the system aims to acquire complex decision-making skills through trial and error. By learning from past experiences and iteratively refining its approach, the system improves its adaptability in two semantically related scenarios. Another objective of the system is to showcase its superiority in terms of learning efficiency, adaptability, and overall performance across a diverse range of tasks and environments.

Furthermore, the system places importance on understanding the A3C algorithm. A3C, or Asynchronous Advantage Actor-Critic, is a state-of-the-art algorithm in the field of deep reinforcement learning. Additionally, the system acknowledges the usefulness of the OpenAI gym environment for deep reinforcement learning. By utilizing the OpenAI gym environment, the system can effectively develop and benchmark its performance, ensuring its compatibility and effectiveness in real-world scenarios.

1.4. Organization of the Study

Together with this chapter, this thesis is separated into five sections. The following chapters of the thesis are arranged as follows:

Chapter 2 describes the background theory and state-of-the-art of Deep Reinforcement Learning and discusses the realization concepts for reinforcement learning agents with the Markov Decision Process (MDP) and the logic behind the policy iteration approach and the value iteration approach.

Chapter 3 explains about the methods used in thesis and gives the concept of the actor and critic for each worker and how to send and get parameters from global network (the supervisor for every worker in training).

Chapter 4 is an empirical study of the DRL agent performance variations when the level of semantic similarity between tasks learned from various gaming contexts (Atari 2600) is determined. The implementation and the description of the test results from six game simulations are deeply detailed, such as training time and data description.

Chapter 5 presents the conclusion drawn from the experiments and offers a few instructions for future studies.

Chapter 2

Background Theory

In this chapter, the background theory of agent and reinforcement learning is introduced, as well as deep learning. Depending on the different types of environments, there will be various rewards, actions, and decisions. The solution to the sequence decision with the Markov Decision Process (MDP) under uncertainty is combining the two problematics of decision and sequential decision under uncertainty are explained. A Markov decision process is a framework that aims to describe the current situation of an agent, incorporating elements such as state, decision (or action), reward, and their impact on the process dynamics.

2.1. Intelligent Agents

An agent is anything that can be viewed as perceiving its environment through sensors acting upon that environment through actuators [4]. An agent perceives some information and specifies the action taken by the agent in its environment. The way to evaluate the performance measure is the behavior of the agent in an environment. Agents that must operate robustly in rapidly changing, unpredictable, or open environments where there is a significant possibility that actions can fail are known as intelligent agents, or sometimes autonomous agents [5].

2.1.1. Abstract Architectures for Intelligence RL Agents

Reinforcement Learning (RL) agents are intelligent systems that learn to make decisions and take actions in dynamic environments to maximize cumulative rewards. In Figure 2.1, these agents operate within an environment, perceive its state, and interact with it through actions. Through a trial-and-error learning process, RL agents aim to discover optimal strategies or policies for maximizing long-term rewards. The core components of an RL agent include the environment, the decision-making policy of the agent, the value function, and the learning algorithm. The environment provides observations and rewards to the agent, while the policy determines the behavior of the agent and action selection. The value function estimates the expected cumulative rewards for different states or state-action pairs, aiding the decision-making process.

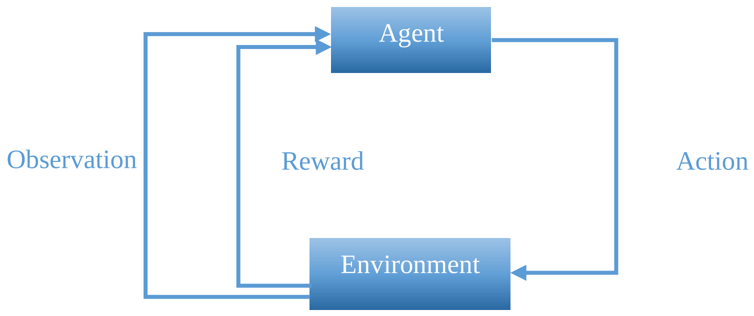


Figure 2.1. Structure of Reinforcement Learning

The learning algorithm updates the policy and value function of agents based on the observed experiences, using techniques such as temporal-difference learning, Monte Carlo methods, or function approximation. RL agents have shown great success in a wide range of domains, including robotics, game playing, recommendation systems, and autonomous vehicles, among others. By combining exploration and exploitation, RL agents continuously adapt and improve their decision-making abilities, making them a powerful tool for autonomous learning and intelligent decision-making in complex and uncertain environments.

2.2. Deep Learning

Deep architectures are made up of various levels of non-linear operations, such as many hidden layers in neural nets or the complex propositional formulas that reuse numerous sub-formulas machine learning algorithms learn a predictive model from a data set; they are linear regression, support vector machines, decision trees, artificial neural networks, random forests, and reinforcement learning. The aims of these methods are to generalize, make predictions, and approximate a function for data from a mathematical point of view. Machine learning has been allowed by deep network and deep learning to be applied to high dimensional, complex, problems, such as recognizing objects localization and classification in high resolution (mega-pixel) images. Deep neural networks consist of many layers that mean more than one hidden layer in between the input and output layer and use different type of connections.

Supervised learning, or learning from a training set of labeled examples, is the field of machine learning that is supplied by a knowledgeable external supervisor. Each example is a description of a situation together with a specification—the label—of the correct action the system should take in that situation, which is often to identify a category in which the situation belongs [7]. The kind of learning aims to generalize or extrapolate but it is not adequate learning with interaction.

Unsupervised learning, which finds structure hidden in the collection of unlabeled data, is quite different from reinforcement learning. Since reinforcement learning does not rely on examples of appropriate conduct, it is tempting to mistake it for unsupervised learning. However, reinforcement learning aims to maximize a reward signal rather than seeking out hidden structure. Finding structure in the experience of an agent can be helpful for reinforcement learning, but it does not solve the issue of maximizing a reward signal on its own.

Deep reinforcement learning explores complex settings in a way that is reminiscent of how kids learn: by trying things out in a playful way, receiving feedback, and trying again. The machine appears to have some features of human learning; artificial intelligence is touched by deep reinforcement learning. Regarding a multi-task environment where various outputs are produced for various activities, all of which are drawn from a common pool of high-level features. Given that many of these learned traits are shared across *m* tasks, statistical power in proportion to *m* is shared. Assume for a moment that these learnt high-level features can be combined with lower-level intermediate features to create their own representation. Once more, statistical strength can be increased in a similar manner, and this tactic can be used to advantage at every level of a deep architecture [6].

2.3. Reinforcement Learning Problem

There are machine learning methods where the only method that does not learn from a dataset is called reinforcement learning. The agent interacts directly with its environment to collect experiences and teach itself from past experiences to get the maximum rewards. Reinforcement Learning differs from Supervised Learning, which is learning from a training set of label examples supported by a knowledgeable external supervisor. Reinforcement Learning differs from Unsupervised Learning, which typically finds structure hidden in collections of unlabeled data as well. The objective of reinforcement learning is to find the best policy: a function that gives the best action for each state that presents the word. The agent is to find the actions in each state that maximizes the long-term accumulated expected rewards. In other words, this optimal function of states is called optimal policy. To obtain a lot of reward, a reinforcement learning agent must prefer actions that it has tried in the past and found to be effective in producing reward [7].

2.3.1. Environment

To illustrate the general idea of reinforcement learning, the experiment needs simulations that plays the big role to get the optimal solutions. An environment is a world for agents to learn and act. In this thesis, it will show off the optimization of deep reinforcement learning method A3C with six various game environments where are Breakout, Pong, Space Invaders, Demon Attack, River Raid, and Adventure.

Breakout environment is with eight rows of bricks. The color order from the top down of bricks is red, orange, green and yellow. The agent must knock down as many bricks as possible by using a single ball and the paddle below to ricochet the ball against the bricks and eliminate them. If the agent paddle misses the ball rebound, the agent will lose a chance. The agent has five chances to clear two screens of bricks. When the agent hits the top-level red bricks score seven points each, orange bricks earn five points, green bricks reward three points and yellow bricks score one point each. The paddle grows smaller to one-half its length after the ball has collapsed through the last row and hit the upper wall. Ball speed increases at specific intervals: after four hits, after twelve hits, and after making contacting with the orange and red rows.

Pong is inspired by table tennis that two players compete each other. The agent controls an in-game paddle by moving up and down to hit and return the ball back to the opponent. The goal is for agent to reach 21 points before the computer; points are earned when one fails to return the ball to the other. The agent earns negative one point when missing to touch the ball.

In the two-dimensional shooter game Space Invaders, the player maneuvers a laser gun horizontally across the bottom of the screen to fire at aliens descended from above. The object of the game is to eliminate six rows of aliens that move horizontally back and forth across the screen as they make their way toward the bottom of the screen. Some versions of the game have various numbers. By firing the laser weapon at the alien, the agent will gain points by killing it. The soundtrack in the game and the aliens movements both quicken as more aliens are slain. After the aliens are defeated, a harder wave follows, and so on in an endless cycle. The agent has got three lives.

As they move toward the bottom of the screen, the aliens fire at the cannon in order to destroy it. The alien invasion is successful if they get it to the bottom, and the game is over. Periodically, a unique "mystery ship" will move over the top of the screen and, if destroyed, will grant extra points. The amount of stationary defense bunkers that are partially shielding the laser cannon changes depending on the edition, and they are eventually destroyed by many blasts from the aliens or agent. The final laser base of the agent must be destroyed in order for terminating the game.

The Demon Attack agent, who is stranded on the ice planet Krybor, utilizes a laser cannon to exterminate legions of approaching demons. Similar to other space-themed shooters, the demons show up visually in waves that merge from the sides of the screen to the area above the player cannon.

The demons use new weaponry with each wave, such as long streaming lasers and laser clusters, to assault. Beginning in Wave 5, demons also split into two more little, avian-like entities that eventually try to descend onto the player cannon. From Wave 9, the shoots of the demons follow exactly beneath the monsters, making it challenging for the player to slink underneath to get in a direct shot.

In River Raid, looking down from above, the spy conducts a raid behind enemy lines by piloting a fighter jet over the River of No Return. However, it can accelerate and decelerate. The aircraft of the agent can only move left and right on the screen. The jet will be crashed if it runs out of fuel, collides with a riverbank or an enemy vessel, or crashes. The amount of playtime is practically limitless if fuel can be refilled and provided the player avoids injury. For firing enemy tankers (30 points), helicopters (60 points), fuel depots (80 points), aircraft (100 points), and bridges, the agent receives points (500 pts). When the airplane passes over a fuel station, it refuels. A gaming level concludes at a bridge. Tanks at the edge of the river that fire at the player aircraft and hot air balloons that score 60 points when shots are added to non-Atari 2600 versions of the game. The checkpoints in the game are also destroyed bridges. If the agent crashes the planes, they will begin their next life near the last bridge that was demolished.

In Adventure game, the Enchanted Chalice was taken by a malicious magician, who then hid it someplace in the Kingdom. Recovering the Enchanted Chalice and putting it within the Golden Castle, where it belongs, is the goal of the game. The Evil Magician has produced three Dragons to obstruct the efforts obtaining the Golden Chalice, so this is no simple assignment. There are Grundle, the Green Dragon, which is vicious and mean; Yorgie, the Yellow Dragon, which is just plain mean, and Rhindle, the Red Dragon, which is the most vicious of them. In addition to being the quickest Dragon, Rhindle is also the hardest to outmaneuver.

There are three castles in the Kingdom; the White Castle, the Black Castle, and the Golden Castle. Each castle has a Gate across the entrance. The Gate can be opened with the corresponding-colored Key. Inside each Castle are rooms (or dungeons, depending on which Skill Level players are playing). The Castles are separated by rooms, pathways, and labyrinths. Common to all the Skill Levels is the Blue Labyrinth through which the Black Castle will be found. Skill Levels 2 and 3 have a more complicated Kingdom.

2.3.2. Reward/Goal

In a reinforcement learning problem, the goal of agent is defined by a reward signal. The environment gives the RL agent a single number for each time step, which is called a reward. Hence the reward signal defines what are the features of the problem faced by the RL agent. The only way the agent can influence the reward signal is through its actions, which can have a direct effect on reward, or an indirect effect through changing the environmental state [7]. A sequence of positive rewards brings about an appreciable policy, and a negative reward gives rise to an undesirable behavior.

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + r_{t+4} + r_{t+5} + \cdots r_T \qquad (2.1)$$

where R_t describes the return (the accumulate sum of rewards), *t* a particular time steps up to *T*. Otherwise, for stochastic environment, the goal is to maximize the expected value of the return with a task.

$$R_t = \sum_{k=0}^{T} r_{t+k+1} \qquad (2.2)$$

A mathematically convenient way to avoid infinite returns and values is defined as a discounted factor. A discounted factor can vary from $0 < \gamma < 1$ and can be considered as a learning parameter. Every future reward at time stamp t is discounted by γ^{k-1}. If γ approaches zero, the agent considers rewards near the present state as much more valuable. On the other hand, a γ is close to one that changes agent to behave greedily and consider future rewards as equally important.

$$R_t = \sum_{k=0}^{T} \gamma^{k-1} r_{t+k+1} \tag{2.3}$$

2.3.3. Actions

Table 2.1. Actions with Specific Number for Adventure and River Raid

No.	Action	No.	Action
0	NOOP	9	DownLeft
1	Fire	10	UpFire
2	Up	11	RightFire
3	Right	12	LeftFire
4	Left	13	DownFire
5	Down	14	UpRightFire
6	UpRight	15	UpLeftFire
7	UpLeft	16	DownRightFire
8	DownRight	17	DownLeftFire

- The numbers of actions of Breakout and Pong are 0 to 3 (NOOP (No Operation), Fire, Right, Left).

- The numbers of actions of Space Invaders are 0 to 5 (NOOP, Fire, Up, Right, Left, Down), and the numbers of actions of Demon Attack is 0 to 5 (NOOP, Up, Right, Left, Down, Upright).

- The numbers of actions in River Raid and Adventure are 0 to 17 (all actions from Table 2.1).

2.4. Markov Decision Processes

Markov decision problems (MDPs) are general mathematical formalisms for representing shortest path problems in stochastic environments [8]. In Figure 2.2, a Markov decision process relies on the notions of state, describing the current situation of the agent, action (or decision), affecting the dynamics of the process, and reward, observed for each transition between states [7].

- ***State(S)***: The set of states that the agent ***experiences*** when interacting with the environment. The states are assumed to have the Markov property.

- ***Action(A)***: The set of legitimate actions that the agent can execute in the environment.

- ***Transition***: Moving from one state to another is called transition.

- ***Transition Probability(P)***: The set of probabilities that the agent will move from one state to another is called transition probability.

- ***Reward(R)***: The reward function that determines what reward the agent will get when it transitions from one state to another using a particular action.

- The Markov Property states that: **"Future is independent of the past given the present"**.

- A Markov decision process is often denoted as **M=⟨S, A, P, R⟩**.

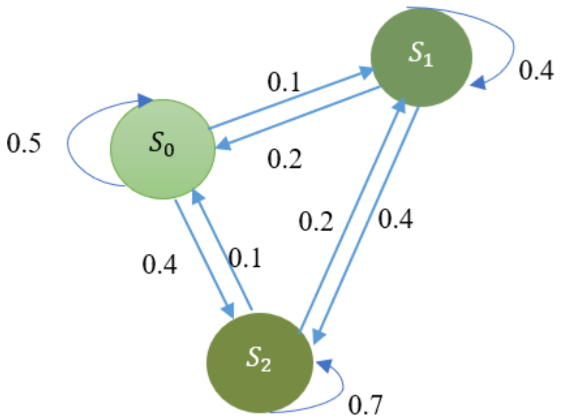


Figure 2.2. Markov Decision Process

Markov Decision Process, where:

- At time step ***t=0***, environment samples initial state $\boldsymbol{s_0 \sim p(s_0)}$

- Then, for ***t=0*** until done:

- Agent selects action $\boldsymbol{a_t}$

  - Environment samples reward $\boldsymbol{r_t \sim R(. | s_t, a_t)}$

    - Environment samples next state $\boldsymbol{s_{t+1}} \sim \boldsymbol{P}(.\,|\boldsymbol{s_t}, \boldsymbol{a_t})$
    - Agent receives reward $\boldsymbol{r_t}$ and next state $\boldsymbol{s_{t+1}}$

- A policy π is a function from *S* to *A* that specifies what action takes in each state.
- Objective: find policy $\boldsymbol{\pi}^*$ that maximizes cumulative discount reward: $\sum_{t\geq 0} \gamma^t r_t$.

2.4.1. Policy Iteration

If the agent sees that one action is better than all other actions, then the exact magnitude of the utilities for the states involved need not be precise. This gives the suggestion which is the alterative way to find optimal policies. The policy gathering the highest amounts of rewards to achieve the specific goal is considered by the optimal policy, denoted by π^*.

Policy Evaluation: find out the value of each state under current policy: Policy π_i-calculate $U_i = U^{\pi_i}$, the utility of each state if π_i were to be executed.

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U^{\pi}(s') \tag{2.4}$$

Policy Improvement: change the action, in each state, to improve value: Calculate a new policy π_{i+1}, using one step look-ahead based on U_i.

$$\pi(s) = \underset{a \epsilon A(s)}{argmax} \sum_{s'} P(s'|s, a) U^{\pi}(s) \tag{2.5}$$

2.4.2. Value Iteration

The value function is to calculate the utility of each state in order to use when the agent decides what the optimal action in each state. If the utility of a state is the expected sum of discounted rewards from that point onwards, then there is a direct relationship between the utility of a state and the utility of its neighbors: *the utility of a state is the immediate reward for that state plus the expected discounted utility of the next state, assuming that the agent chooses the optimal action* [4]. On the other hand, the *n* equations contain *n* unknowns—the utilities of each state—and there are one of *n* possible states. A system of non-linear equations cannot be solved with linear operation techniques; there is only one thing to try: an iteration approach.

$$V: V^{\pi} \rightarrow \mathbb{R} \qquad V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\{\textstyle\sum_{i=0}^{\infty} \gamma^i r_{t+i+1|s_t=s}\} \tag{2.6}$$

As the expected value evolves from experience samples within the task, a value function V^{π} is estimated by “trail-and-error”.

$$V = R + \gamma P v \tag{2.7}$$

where *V* is a column vector with one entry per state

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1 \\ \vdots \\ \mathcal{R}_n \end{bmatrix} + \gamma \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{11} & \dots & \mathcal{P}_{nn} \end{bmatrix} \begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} \tag{2.8}$$

2.4.3. Asynchronous Advantage Actor-Critic (A3C(On-Policy))

The algorithm, which is called Asynchronous Advantage Actor-Critic (A3C), maintains a policy $\pi\ (a_t | s_t;\ \theta)$ and an estimate value function $V\ (s_t;\ \theta_v)$. Like variant of n-step Q-learning, variant of actor-critic also operates in the forward view and uses the same mix of n-step returns to update both the policy and the value-function. The policy and the value function are updated after every t_{max} actions or when a terminal state is reached. The update performed by the algorithm can be seen as $\nabla_{\theta'} log\pi(a_t | s_t; \theta') A(s_t, a_t, \theta, \theta_v)$ where $A(s_t, a_t, \theta, \theta_v)$ is an estimate function given by $\sum_{i=0}^{k-1} \gamma^i r_{t+1} + \gamma^k V(s_{t+k}; \theta_v) - V(s_t; \theta_v)$ where k can vary from state to state and is upper-bounded, by t_{max} [1]. A3C is an enhancement of the actor-critic algorithm, where the actor module could be treated as a function approximator unit that governs agents at each state, and the critic module acts as a sort of judging unit that effectively evaluates the effectiveness of the policy created by the actor and then provides feedback.

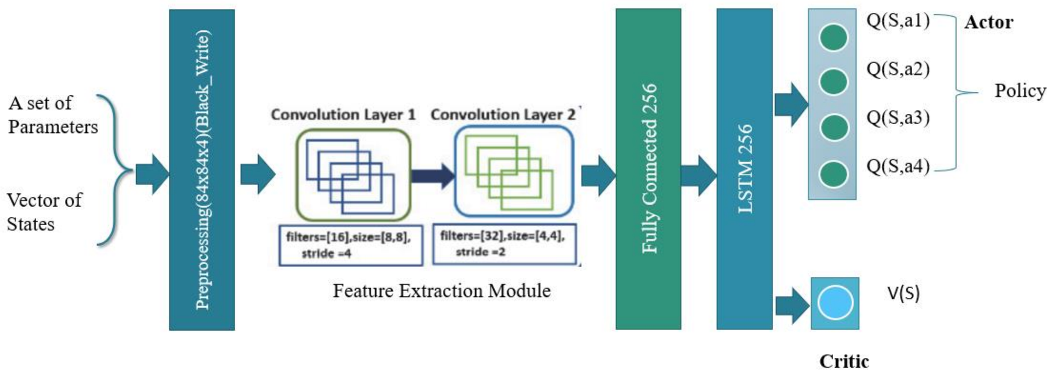


Figure 2.3. Architecture of A3C

2.5. Exploration vs Exploitation

In reinforcement learning, the agents are always faced with the exploitation dilemma and the exploration dilemma, which have been hardly studied by mathematicians for many decades. Exploration could lead to long-term benefits and allow the agent to build on its knowledge about each action and each state. To obtain a lot of rewards, exploitation chooses the greedy approach that the agent has tried the actions from the past and found to be effective in producing reward. The agent, however, is being greedy because it is using the predicted value rather than the real value, so there is a potential it will not receive the highest payout. It is necessary for an agent firstly to investigate the most states, and then, when the agent is confident enough to have thoroughly examined the world, it utilizes the knowledge obtained to acquire superior results.

2.5.1. Epsilon-Greedy Strategy Selection

E-greedy is a common strategy used in reinforcement learning to balance exploration and exploitation. In the e-greedy algorithm, the agent chooses a random action with a certain probability (called the exploration rate or epsilon) and chooses the action with the highest expected reward with the remaining probability.

The idea behind the e-greedy algorithm is that by occasionally choosing a random action, the agent can explore its environment and learn more about the possible actions and their rewards. This can help the agent to improve its policy over time and make better decisions. At the same time, by choosing the action with the highest expected reward most of the time, the agent can exploit its current knowledge of the environment and maximize its rewards.

One of the key advantages of the e-greedy algorithm is its simplicity. It is easy to implement and can be applied to a wide range of RL problems. However, the e-greedy algorithm also has some disadvantages. For example, it can be less effective than more sophisticated exploration strategies in some cases. Additionally, because it always chooses a random action with a certain probability, it can be slow to converge of an optimal policy in some situations.

Overall, the e-greedy algorithm is a widely used and effective approach to exploration in reinforcement learning. It is particularly well-suited to problems which the balance between exploration and exploitation may vary over time, and a simple and flexible approach is desired. For adjusting between exploration and exploitation, the agent picks a random action at each time step with fixed probability with e-greedy, $0 < \varepsilon < 1$, rather than picking the greedy one of the learned the optimal actions. Epsilon-greedy algorithm exploits with the probability *1-ε* as well as explores with the probability ε.

2.6. On-Policy vs Off-Policy Method

While Off-Policy methods choose a separate policy to update the same action-value function, On-Policy methods use the action selected to update the action-value function on the same policy. On other words, on-policy tends to evaluate the policy that is employed when making decisions and off-policy tends to improve a policy different from that which was utilized to produce the data. If the agent has to manage the optimization of the exploration, on-policy reinforcement learning is helpful. Off-policy reinforcement learning might be a better choice when the agent does not need to explore much. Off-policy learning deployed in the real world can be rather cost-effective because of the propensity to experiment, seeking out novel approaches, and preparing for future reward tasks.

2.7. Model-Based vs Model-Free Approach

When using model-based reinforcement learning, a world model is created. According to computer science, it is a transition probability, which describes how the world can change depending on the type of action that is taken in it. The agent must do more than just implement a model of the environment in order to qualify as model-based. A mental simulation will be performed when it is in a specific situation where it has already learned the model of the environment. It would essentially be searched through the mental model built to determine what kind of results a specific set of activities would produce. And once the course of action is identified that will lead to the desired outcome, it is begun to carry out physically those acts. In other words, the agent must forecast potential benefits linked to various acts. The development of AI systems that can master board games like Chess and Go, where the environment is deterministic, has been particularly successful using model-based reinforcement learning.

In model-free approach, the "law of effect" argues and responses that has negative consequences become less likely to occur in the future and actions that have good effects in a scenario become more likely to occur again in that situation. The foundation of model-free reinforcement learning is the law of effect. An agent senses the environment, acts, and measures the reward in model-free reinforcement learning. Typically, the agent begins by performing random behaviors before eventually repeating those that are linked to greater rewards. Model-free reinforcement learning lacks any direct world knowledge or world models. Trial and error must be used by the RL agent to feel directly every result of every action.

Model-free reinforcement learning is a type of machine learning algorithm that enables an agent to learn from its interactions with the environment. In contrast to model-based RL, where the agent uses a model of the environment to make predictions and plan its actions, model-free RL algorithms do not rely on an explicit model of the environment. Instead, they learn directly from the raw sensory input and the rewards they receive from the environment. It has been challenging to train an agent using model-free reinforcement learning to solve control tasks directly from high-dimensional images, and using reconstruction loss in an off-policy learning algorithm frequently results in training instability [9].

Some of the key advantages of model-free RL include its simplicity and flexibility, as it does not require building and maintaining a model of the environment. This can make it easier to apply to real-world problems, where the environment may be complex and hard to model accurately. Additionally, because the algorithm does not rely on a model of the environment, it can adapt in the environment more quickly and easily than model-based approaches.

However, the lack of an explicit model of the environment can also be a disadvantage of model-free RL. For example, it can make it more difficult to understand why the agent is making the decisions it is making, and it can make it harder to evaluate the performance of the algorithm. Additionally, because the algorithm relies on learning directly from experience, it may require more data and more computational resources to train than model-based approaches.

Overall, model-free RL is a powerful and widely-used approach to reinforcement learning that can be applied to a wide range of problems. It is particularly well-suited to problems where the environment is complex and hard to model accurately and where the ability to adapt quickly to the changes in the environment is important.

2.8. Transfer Learning Approach for Reinforcement Learning

The majority of deep reinforcement learning algorithms have a limited range of applications since they are data inefficient in complex and rich contexts. Multitask learning with shared neural network parameters is one approach to increasing data efficiency, as efficiency may be increased by transferring across related tasks. This approach tries to incorporate transfer learning support into reinforcement learning [10]. The primary focus of transfer in reinforcement learning is on developing strategies to transfer knowledge from a collection of source tasks to a target task, and this experiment produced promising results when the levels of similarity between the source and target tasks were similar. The underlying learning algorithm can use the transferred knowledge to solve the target task effectively if the degree of similarity between the source and target tasks is sufficiently high [11].

Various transfer methods, such as instance transfer, representation transfer, or parameter transfer, are used to transmit knowledge in different ways between the source and target activities. Each of these techniques uses underlying transfer algorithms that significantly rely on prior knowledge acquired through the completion of a collection of related source tasks and then utilize that knowledge as a guide to slant learning on any new assignment [12].

In multi-agent systems, where agents interact with other agents acting in the same environment and subsequently exploit the knowledge gained from their activities as well, there have been research attempts to extend the same approaches. Multi-agent systems often build their initial policies for the agents in the target task using the joint policies that the agents in the source task (training task) learned [13]. A long-standing objective of machine learning is continuous process, where agents are able to transfer knowledge from prior tasks to speed up convergence in addition to learning (and remembering) a sequence of tasks. These requirements are directly incorporated into the model architecture of progressive networks, which prevent catastrophic forgetting by creating a new neural network (a column) for each task being solved and by transferring lateral connections to the features of previously learned columns [14].

Without compromising training stability or data efficiency, Importance Weighted Actor-Learner Architecture (IMPALA) can scale to thousands of machines. IMPALA actors communicate experience trajectories (sequences of states, actions, and rewards) to a centralised learner as opposed to the popular A3C-based agents, in which workers communicate gradients with respect to the policy parameters to a central parameter server. IMPALA employs a GPU to update mini-batches of trajectories while aggressively parallelizing all time independent operations because the learner in IMPALA has access to full trajectories of experience [15]. The method is the same for single-game learning as it is for multi-task Deep Q Learning (DQN): the network is optimized to forecast the average discounted return of each potential action given a sparse set of consecutive observations. The game is switched every episode, distinct replay memory buffers are kept for each task, and training is evenly spaced out between all tasks, just like in multi-task distillation. As in multi-task DQN, the game label is used to toggle between various output levels, enabling a separate output layer, or controller, for each game. The multi-task DQN loss function is still equivalent to single-task learning under this architecture [16].

2.9. Chapter Summary

This chapter has examined how the agent given only its occasional rewards and percepts explores in an unknown environment. For the whole AI problem, the type of information that must be learned is determined by the overall agent design. Reinforcement learning using an approximate functional representation to generalize over state is one of the most active areas of machine learning research area for handling high-dimensional, continuous, partially observable environments.

Chapter 3

Optimization Reinforcement Learning from Distinct Perspectives

The asynchronous deep reinforcement learning strategy, which has been shown to be a more effective method in terms of performance, generatability to both on/off policy schemes, and computational efficiency. It turns out that including multiple different game environments within this framework is natural and effective: perform knowledge transfer by sharing model parameters across different tasks, achieve scalable multi-task learning by launching multiple threads to explore concurrently different environments, and update model parameters asynchronously [17].

The essential feature of the A3C method is its capacity to learn simultaneously numerous instances of a single target task and to enhance the performance of the model by sharing knowledge across multiple instantiations. The problems of a RL agent with efficient exploration, a sufficient number of training samples, and the training time necessary to achieve an optimal performance level would be mitigated by having multiple actor-critic models running concurrently across two semantically similar environments, is called as Hybrid.

By transferring learning across the operational environment of the agent, numerous environments with a high level of semantic similarity will indirectly increase the partial observability. However, this knowledge transfer may have a beneficial or negative effect on overall learning progress and performance. The transferred information could have a detrimental effect if the source tasks and the target tasks differ significantly. The primary concept of knowledge transfer learning in a context of multiple tasks is that sharing knowledge amassed via learning from a collection of source samples under the supervision of one task agent may enhance the performance of another task agent while learning on the target task. While the policy gradient theorem suggests that it is preferable choosing a parameterization for the actor, after which compatible features for the critic can be determined, many actor-critic algorithms use the exact same set of features for both the actor and the critic [18].

This thesis shows the A3C algorithm is able to improve the model performance caused by knowledge transfer, is also able to learn both tasks simultaneously when there is semantic similarity between works. Multi-Task Reinforcement Learning is inspired by cognitive capabilities of the brain, in particular the ability to adapt and leverage knowledge efficiently in multiple complex contexts, an important feature in many areas such as data analysis, robotics, and medicine.

3.1. OpenAI

The OpenAI gym is a place where learning agents can be created and tested. Although it is concentrated and most appropriate for reinforcement learning agents, it does not prevent one from experimenting with alternative techniques, such as hard-coded game solvers or other deep learning techniques. If the research community has a good benchmark, the researchers can easily compare algorithms with built on recent progress in Reinforcement learning. As a result, a lot of benchmarks have been produced, such as Arcade Learning Environment that exposed reinforcement learning problem, or a collection of Atari games as well as recently the RLLab benchmark for continuous control.

With the experience of the agent divided into a number of episodes, OpenAI Gym focuses on the episodic setting of reinforcement learning. Each episode starts with a randomly selected initial state for the agent, and from there, interaction continues until the environment reaches a terminal state. Achieving a high level of performance in the least number of episodes is the aim of episodic reinforcement learning, which aims to maximize the anticipation of total reward per episode [19]. The goal of OpenAI Gym is to incorporate the greatest features of these earlier benchmark collections into a software solution that is as convenient and usable as possible. With a common interface, it consists of a variety of jobs (referred to as environments), and this collection will expand over time.

3.2. Preprocessing

The screenshots of game collected from OpenAI are much bigger requirement with a lot of higher resolution than preprocess requirement before the frames input into the neural network. By cropping, the images turn into which the important area is displayed. After all, the image is changed into grayscale and in the end, the image is converted using cv2 with nearest-neighbor interpolation and next change the frame datatype into np.unit8.

The position will be known on behalf of the agent for a while looking at the image. In view of the fact that only one timestep will be considered, insufficient information cannot be used accurately to determine some situations. When presented with two timesteps, the ball speed can be determined; when presented with three timesteps, the ball acceleration can be determined; and so on, for example, in pong and breakout games. Current state at time *t* in game is a stack of the images at *t, t-1, t-2, t-3*. If the most recent frames are heap together, when the size of the images is set to be (84, 84), the shape of the state at time *t* will be (84, 84, 4).

3.3. Convolutional Neural Network (CNN)

A subtype of Artificial Neural Network (ANN) that has deep feed-forward architecture and amazing generalizing ability in comparison to other networks with fully-connected (FC) layer. In contrast to other networks with FC layers, the Convolutional Neural Network (CNN), also known as ConvNet. It can learn highly abstracted features of objects, particularly spatial data, and can identify them more effectively. A deep CNN model is made up of a limited number of processing layers that can learn different input data features at different levels of abstraction. The deeper layers learn and extract the low-level features, whereas the initiatory layers learn and extract the high level features (with less abstraction) (with higher abstraction) [20]. It can learn highly abstracted features of objects, particularly spatial data, and can identify them more effectively.

- The feature extraction layers, and the classification layer collaborate to learn in CNN, which improves the output organization of the model and increases its reliance on the extracted features.

- Using the other types of neural network instead of the convolutional neural network makes more difficult to create a large network.

- The case of the weight sharing feature of CNN which bring down the number of trainable parameters in the network and another main reason is that assisted the model to keep away overfitting and to improve generalization also.

3.3.1. Kernel

Convolutional layer is the most important component of any CNN architecture. It contains a set of convolutional kernels (also called filters), which gets convolved with the input image (N-dimensional metrics) to generate an output feature map [20]. A kernel can be described as a grid of discrete values or numbers, where each value is known as the weight of this kernel. All the weights of a kernel are allocated with random numbers during the starting of training process of an CNN mode. These kernels often have a low spatial dimension yet cover the entire depth of the input. Each filter is convolved across the spatial dimensions of the input by the convolutional layer as the data reaches it, creating a 2D activation map [21].

3.3.2. Convolutional Operation

Unlike the input format of other classical neural network, the input in CNN is a multi-channeled image (RGB-(Red-Green-Blue) is 3 channeled image and Gray-scale image is a single channeled). In Figure 3.1, the convolution operation goal is to take the high-level characteristics of the input images: like edges and extract them. ConvNets do not have just one convolutional layer. Typically, the first ConvLayer is in charge of capturing the Low-Level features like edges, color, gradient orientation, etc. With more layers, the architecture adjusts to High-Level characteristics as well, providing us a network with a comprehensive grasp of the images in the dataset, just like human being would. The procedure yields two different types of results: one where the dimensionality of the convolved feature is decreased in comparison to the input, and the other where it is either increased or stays the same.

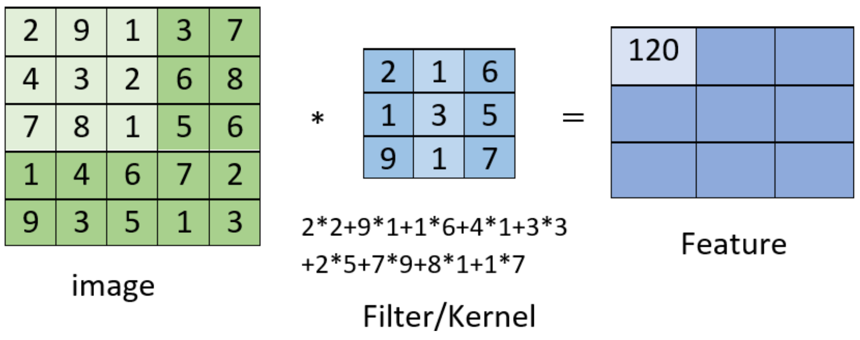


Figure 3.1. Operation of Convolution

3.3.3. Activation Functions

The activation function in neural network is charge for turning the summed weight input to the activation of the node. The Rectified Linear Activation Function or ReLU is a linear function which outputs the input directly when it is positive, if not, it outputs zero. The reason why ReLU is the default activation function for a lot of neural networks is that a model is simpler to train and perform frequently better. When implementing multiple perceptron and convolutional neural networks, activation functions are usually utilized.

3.3.4. Fully Connected Layer (FC)

Adding a Fully-Connected layer behind CNN is an (usually) unexpensive way of learning non-linear combinations with the high-level features as represented by the output of the convolutional layer. The FC layer is learning a possibly non-linear function in that space. Now that the input image is converted into a suitable form for Multi-Level Perception (MLP), flatten the image into a column vector as displayed in Figure 3.2. The flattened output is fed to a feed-forward neural network and backpropagation applied to every iteration of training. Over a series of epochs, the model is able to distinguish between certain low-level features and dominating in images and classify them using the Softmax Classification technique.

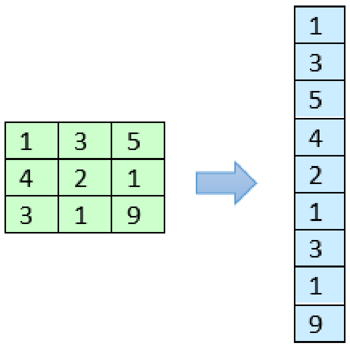


Figure 3.2. Matrix to Vector

3.4. Implementation of LSTM

Since there may be lags of uncertain length between significant occurrences in a time series, LSTM networks are well-suited in categorizing, processing, and making predictions based on time series data. To solve the vanishing gradient issue that can arise when training conventional RNNs, LSTMs were created. The recurrent hidden layer of the LSTM has unique components known as memory blocks. In addition to specific multiplicative units called gates that regulate the flow of information, the memory blocks also include memory cells with self-connections that store the temporal state of the network. The main innovation in the LSTM design was the inclusion of nonlinear, data-dependent controls in the RNN cell, which can be trained to make sure that the gradient of the objective function with respect to the state signal (the amount directly proportional to the parameter updates computed during training by Gradient Descent) does not vanish [22].

Sequential data often exhibits temporal dependencies, where the current value depends on previous values in the sequence. LSTMs excel at modeling such dependencies by capturing the relationships between past and current states. This makes them powerful in tasks such as time series prediction, where the accurate temporal dependencies modeling is critical. LSTMs offer significant advantages in capturing long-term dependencies, mitigating the vanishing gradient problem, handling variable-length sequences, modeling temporal dependencies, and effective memory management. These features make LSTMs particularly valuable in tasks involving sequential data, where understanding and leveraging temporal relationships are essential for accurate predictions or decision-making.

3.4.1. The Problem of Short-Term Memory

Short-term memory is a problem for recurrent neural networks. They will struggle to transfer information from earlier time steps to later ones if a sequence is too protracted. RNN may exclude crucial information from the initial processing of a paragraph of text to make predictions.

The vanishing gradient issue affects recurrent neural networks during back propagation. The neural network weights are updated using gradients as values. The vanishing gradient problem occurs when the gradient contracts as it travels back in time. A gradient value does not significantly contribute to learning if it becomes very small.

So, layers that get a small gradient update stops learning in recurrent neural networks. Those are usually the earlier layers. Because these layers do not learn, RNN can forget what it seen in longer sequences, thus having a short-term memory.

3.4.2. LSTM

LSTM is a solution for the problem of short-term memory and consists of internal mechanisms called gates, which can supervise the flow of information. These gates are capable of learning which data in a sequence should be kept or ignored. In order to create predictions, it can convey pertinent information along the extensive chain of sequences, as depicted in Figure 3.3. The cell state as the “memory” of the network acts as the transport gate that carries down the sequence chain relative information.

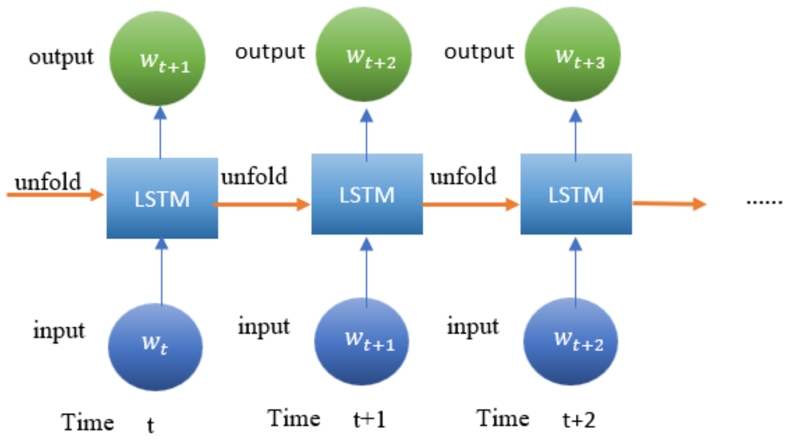


Figure 3.3. Architecture of LSTM

3.4.3. Activation Functions for LSTM Gates

In tanh activation, the values moving across the network are controlled with the aid of the tanh activation. Values are always compressed by the tanh function to fall between -1 and 1.

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}} \quad (3.1)$$

Vectors go through a lot of changes when they pass through a neural network because of various math operations. For example, consider a value that is multiplied by, say, 3, forever. It is easy to see how some values might grow exponentially and become astronomical, making other values appear small. For example, tuple v= [5, 0.01, -0.5] to ([15,0.03, -1.5], [45,0.09, -4.5], [135,0.27,13.5], [405,0.81,40.5], etc..). The output of the neural network is controlled by the tanh function, which ensures that the values remain between -1 and 1. This can be seen in next sentence how the same values from above continue to fall within the tanh function is permitted boundaries. For tuple v= [5, 0.01, -

0.5] changes to ([1, 0.029, -0.9], [0.99,0.09, -0.99], [0.99, 0.26, -0.99], [0.99,0.65, -0.99], etc..).

In the sigmoid function, there are sigmoid activations in gates. Similar to a tanh activation is a sigmoid activation. It compresses data between 0 and 1 instead of compressing values between -1 and 1. This is useful for updating or "forgetting" data because every number multiplied by 0 is 0, which causes values to vanish or be "forgotten". Any number multiplied by one has the same value, hence that value is "preserved" or remains the same. The network can learn which data is crucial to maintain and which should be deleted based on importance.

$$s(x) = \frac{1}{1+e^{-x}} \tag{3.2}$$

3.4.4. Forget Gate

The forget gate decides what information should be kept or thrown away. The sigmoid function processes data from the previous hidden state as well as data from the current input. There are values between 0 and 1. To forget means getting closer to 0, and to keep means getting closer to 1.

3.4.5. Input Gate

Information from the previous hidden state and the current input passes into a sigmoid function, which transforms the values so that they are between 0 and 1, then determines which transforms the data so that 0 implies the value is not important and 1 means the value is important to determine which values will be updated. In order to help the network maintains stability, the hidden state and current input are also provided into the tanh function. The tanh output is then multiplied by the sigmoid output, and which information from the tanh output should be retained will be determined by the sigmoid output.

3.4.6. Output Gate

The next concealed state would be determined by the output gate. Keeping this in mind, the concealed state comprises data about prior inputs. For forecasts, the concealed state is also utilised. It is started by feeding a sigmoid function the current input as well as the prior concealed state. Then, the tanh function is provided the newly updated cell state. The information the hidden state should contain is determined by dividing the tanh output by the sigmoid output. The concealed state is what is produced. Following that, the new cell state and new concealed are carried over to the following time step.

3.5. Optimization

The goal of all optimizers is to find the optimum values or to reach the global minima where the global minimum or the point at which the cost function is at its lowest point. In the beginning of a neural network training, the cost might be high, but the task is to find a gradient each time and update the values of biases and weights. As a result, the cost moves closer and closer to global minimum values. These cost functions are frequently non-convex, that is, it is possible to become trapped in a local minima and have loss never reach the global minimum value. There are ways to prevent the network from becoming convergent, such as changing the rate of learning or using momentum.

3.5.1. Learning Rate

The most crucial component of gradient descent, as well as other optimizers, is probability learning rate. If the learning rate is thought such a step to reach global minima. If a high value is selected for the learning rate, will be drastically altering the weights and bias values, which means making massive jumps to the global minima. As long as a huge probability, it might be overshoot the real minima and end up with local minima. A small learning rate reduces the chance of overshooting minima, but it takes longer for the algorithm to converge, because it takes fewer steps overall but more steps overall. Consequently, it would be necessary to train for a longer amount of time.

3.5.2. RMSProp Optimizer

One of the challenging aspects of deep learning is the optimization of the training criteria over millions of parameters. The difficulty comes from both the size of these neural networks and the fact that the training objective is non-convex in the parameters. It is possible to create better estimators, nevertheless, by comprehending how RMSProp calculates the quantity of interest and the bias it introduces. RMSProp is equivalent to the gradient descent algorithm with momentum. The oscillations are restricted by the

RMSProp optimizer in the vertical direction; hence, the learning rate can be increased, and the algorithm was able to converge quickly in the horizontal direction with a greater increment. Typically, beta is set to 0.9 to represent the momentum value.

$$v_{dw} = \beta \cdot v_{dw} + (1 - \beta) \cdot dw^2 \tag{3.3}$$

$$v_{db} = \beta \cdot v_{dw} + (1 - \beta) \cdot db^2 \tag{3.4}$$

$$W = W - \alpha \cdot \frac{dw}{\sqrt{v_{dw}} + \epsilon} \tag{3.5}$$

$$b = b - \alpha \cdot \frac{db}{\sqrt{v_{db}} + \epsilon} \tag{3.6}$$

v_{dw} value occasionally may be extremely close to zero. Weight value might then skyrocket. The denominator has a parameter called epsilon(ϵ) that is set to a low value in order to prevent the gradients from blowing out.

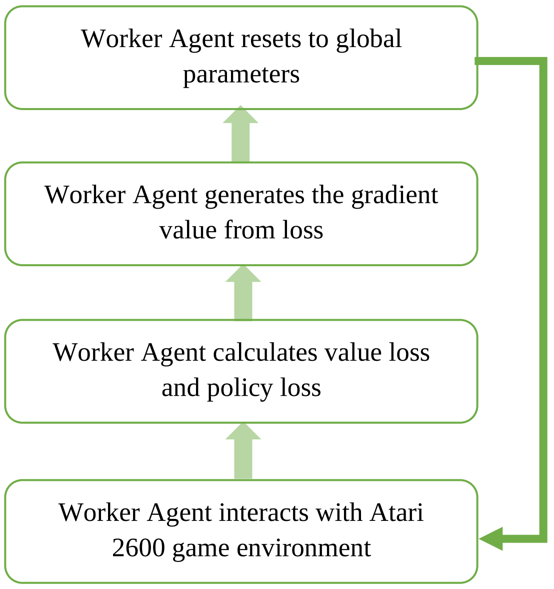
3.6. Multi-Task Deep Reinforcement Learning

With the tremendous growth that has occurred in the AI and DL fields, DRL has established itself as the cutting-edge solution of choice for many benchmark projects and real-world problems. Methods for DRL optimization have attracted a lot of interest and attention as a direct result of this. The development of algorithms is the capable of outperforming human performance on particular tasks that has advanced significantly in the reinforcement learning sector. Most of the time, these algorithms are trained on a single task at a time, with each new work needing the training of a fresh agent instance. As a result, while the learning algorithm is generic, the solutions are not; each agent can only complete the tasks that it was taught how to complete.

In this thesis, the challenge of simultaneously mastering many sequential decision problems is investigated. Finding a balance between the demands of several tasks competing for the finite resources of one learning system is a general problem with multi-task learning. Certain tasks in the collection of learning algorithms can cause them to become sidetracked. Asynchronous standard reinforcement learning algorithms prove that most asynchronous techniques can successfully train neural network controllers due to the stabilizing impact of parallel actor-learners.

3.7. Asynchronous Advantage Actor-Critic (A3C)

A multitask learning strategy based on simultaneous, parallel learning is named by A3C. This means that various intelligent agents, also known as workers, will be functioning in parallel within various instances of the same operating environment. A global value function will be updated asynchronously by all of the workers that are active in the environment. The fundamental idea behind this method is that each of these separate agents would be in a different condition inside the environment at the time of training, at every given time stamp *t*. They can learn independently and in a distinctive way because of this property. The impact of this unique A3C algorithm will be in a position to deliver each of the agents with a very highly efficient learning trajectory within the vast state space of the operating environment. Each worker thread consists of a separate actor-critic pair that simultaneously interacts the surrounding environment. Other name of the agent is the worker, many workers are taught simultaneously and frequently update a global network [23]. The asynchronous aspect arises because the updates do not take place at the same time.


Worker Agent resets to global parameters
Worker Agent generates the gradient value from loss
Worker Agent calculates value loss and policy loss
Worker Agent interacts with Atari 2600 game environment
Figure 3.4. Training Workflow of Worker Agent Thread

3.7.1. Actor-Critic Methodology

Advantage actor-critic algorithm is the combination of policy-based reinforcement learning algorithm and value-based reinforcement learning algorithm. The actor-critic (AC) methodology, in contrast to some simpler techniques that are based on either value-iteration (Q-learning) methods or policy-gradient (PG) methods, combines the best aspects of both methods, namely the algorithms that predict both the value function *V(s)* and the optimal policy function *π(s)*. The policy (a probability distribution of actions) that maps input states to output actions is directly learned by policy-based agents. Based on the expected value of the input state or action, value-based algorithms learn to choose actions. The actor and the critic networks collaborate to solve a specific problem in the actor critic algorithm [18].

The Advantage Function, in its most basic form, determines the agent Temporal Difference (TD) Error or Prediction Error. At each time step, the actor network selects an action, while the critic network evaluates the quality or Q-value of a specific input state. The actor uses this knowledge to instruct the agent to seek out good states and avoid bad states as the critic network learns whether states are better or worse. In advantage function or Temporal Difference Learning (TDL), agents gain knowledge by forecasting future rewards and adapting their behavior in response to prediction mistake. Prediction mistake appears to be one of the ways that the brain learns new things, which is one of the reasons that Temporal Difference Learning is particularly fascinating.

If the step is not the last,

$$TD\_target = R + V(s') \tag{3.7}$$

The TD_target last step,

$$TD\_target = R \tag{3.8}$$

3.7.2. Actor

The actor serves as a policy network that determines whether action a should be executed for a given state *s* at each subsequent time step *t*. In Figure 3.5, the actor module takes the input as the state and gives the output as the best action. A policy-based DRL agent actions within an environment are under the direction of an actor. The goal of a function approximator actor, such as a neural network, is frequently to determine the optimum course of action while a DRL agent is in a state S_t at time step *t*. The critic advice is taken into consideration as the actor module modifies the policy parameters for $\pi_\theta(a|s)$.

$$\pi(a|s, \theta p) = NeuralNetwork(input: s, weights: \theta p) \tag{3.9}$$

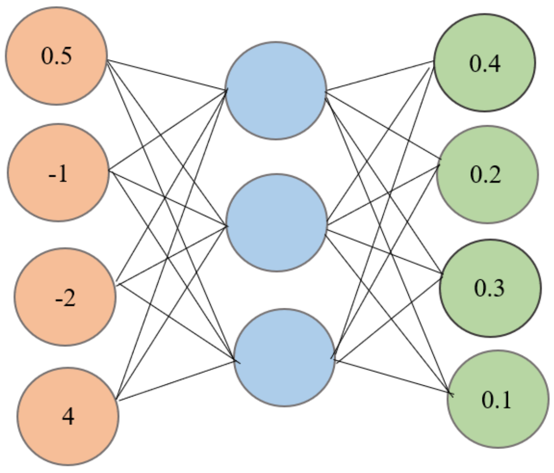


Figure 3.5. Actor Network of Breakout Agent with 4 Actions

3.7.3. Critic

The critic is made up of a value network $V_\pi(s, a)$ that expresses how actionable things are in the given state *s*. The role of critic is to evaluate the action by computing value-based function and to tell how the action taken by the agent will be good. In Figure 3.6, the critic could be implemented with a neural network as a function approximator. Critic generates an evaluation value *V(s, a)* for the actor, who can then modify its strategy for improved outcomes [24]. The value function parameters *w* are updated by the critic module, and depending on the algorithm, they may be either action-value $Q_w(a|s)$ or state-value $V_w(s)$.

$$V(s, \theta v) = Neural\ Network(input: s, weights: \theta v) \tag{3.10}$$

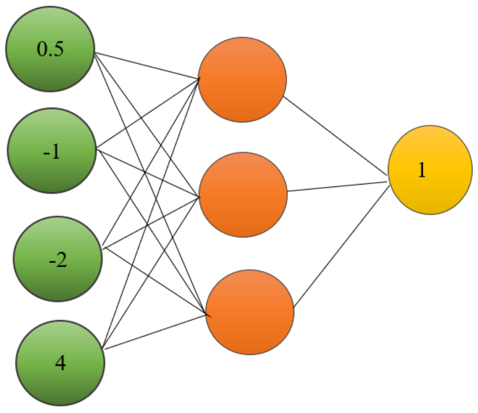


Figure 3.6. The Value of State from the Critic Network

3.7.4. A Worker from A3C

A3C is asynchronous, which means that several actor-critic threads are active at once, each with its own environment. Each thread synchronizes its local parameters with the global parameters every local steps or when a terminal state is reached. According to Figure 3.7, each thread also computes gradients and applies them upstream to the global network. Each thread navigates its environment using a local (CNN and LSTM) while regularly updating a global (CNN and LSTM) that is shared by all networks and has a uniform architecture.

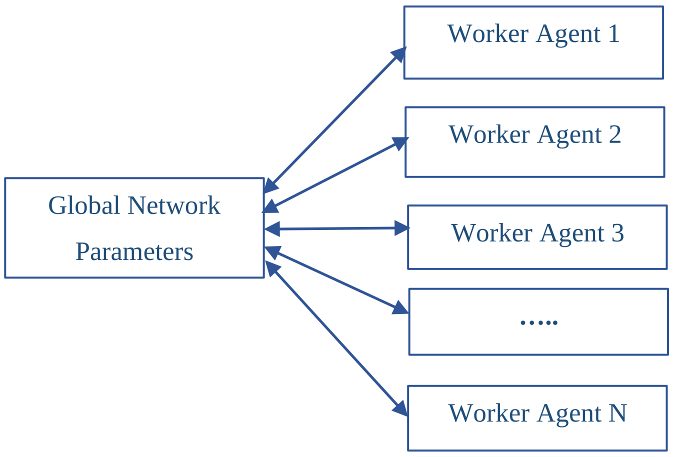


Figure 3.7. Abstract Architecture of Relation between Global and Workers

According to experimental results, A3C is not overfit to state trajectory of any given worker. The agent also learns how much greater the rewards were than its expectations by substituting the value of advantage. The concept of Advantage *A* is used to quantify the discrepancy between expected and actual rewards. The agent gains newfound understanding of the environment as a result, which improves the learning process.

$$Advantage: A = Q(s,a) - V(s) \tag{3.11}$$

where *Q* is the *Q* value that the critic module determined after an actor took a policy-based decision-based action based on the real reward and *TD* mistake. Future rewards accumulated to time t_{max}, or the terminal state, must be discounted using the advantage function $A(S_t, a_t; \theta, \theta_t)$.

$$A(S_t, a_t; \theta, \theta_t) = \sum_{j=0}^{m-1} \gamma^j r_{t+1} + \gamma^m V(s_{t+m}; \theta_v) - V(s_t; \theta_v) \tag{3.12}$$

The following equations, which are derived by adding up all the states in the previous t local executions of each worker agent thread, respectively denote gradients connected to both policy and value networks.

$$\nabla_{\theta'} log\pi(a_t|s_t;\theta)A(s_t,a_t;\theta,\theta_v) \tag{3.13}$$

$$d\theta = d\theta + \partial\left(R - V(s_t;\theta'_v)\right)^2/\partial\theta' \tag{3.14}$$

$$\nabla_{\theta'} log\pi(a_t|s_t;\theta')(R - V(s_t;\theta_v)) + \beta\nabla_{\theta'} H(\pi(s_t;\theta')) \tag{3.15}$$

where β regulates the strength of the entropy regularization term. The hyperparameter H is the entropy.

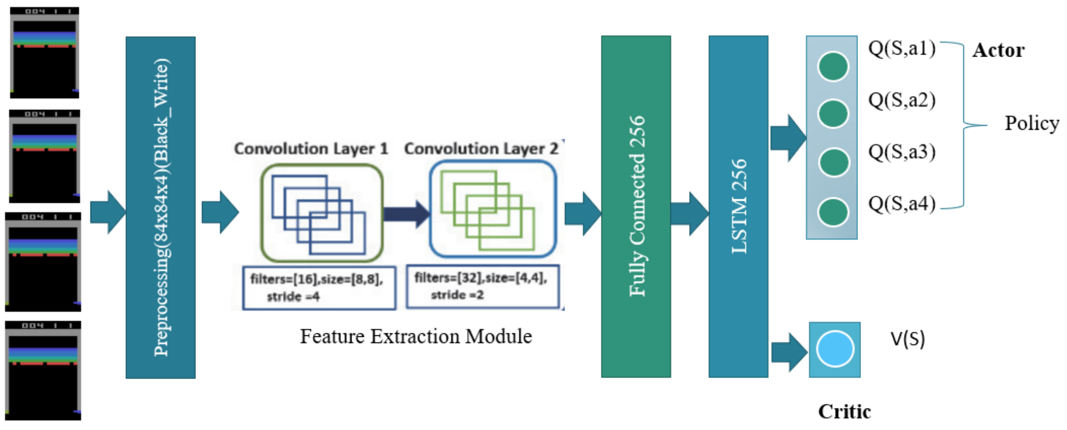

Preprocessing(84x84x4)(Black_Write)
Convolution Layer 1
Convolution Layer 2
filters=[16],size=[8,8], stride =4
filters=[32],size=[4,4], stride =2
Feature Extraction Module
Fully Connected 256
LSTM 256
Q(S,a1)
Q(S,a2)
Q(S,a3)
Q(S,a4)
Actor
Policy
V(S)
Critic

Figure 3.8. Network Structure of Worker Agent

3.8. Hybrid A3C Agents

Asynchronous multi-threaded versions of the advantage actor-critic method is used in the hybrid A3C model. Finding a mechanism to train deep neural network policies reliably and without using a lot of resources is the main goal of this construction model. When trying to improve a DRL agent performance, obstacles including partial observability, the amount of training time and training data samples needed, and efficient exploration become frequently bottlenecks.

The fundamental actor-critic model is extended to two distinct settings with a high degree of semantic similarity in the suggested technique, known as the hybrid A3C model, in an effort to address most of these issues. The essential feature of the A3C method is its capacity to learn simultaneously numerous instances of a single target task and to enhance the performance of the model by sharing knowledge across multiple instances.

In Figure 3.9, the hybrid approach will be heavily relying on the applicability of the multi-threaded capability of the A3C algorithm across semantically related tasks running in two different environments [25]. The key aspect of the hybrid approach is only to enhance the performance of the RL agent through a joint-learning through multi-task learning approach by using deep reinforcement learning. To maximize this diversity, it is possible to implement various exploration rules at the level of a single actor-learner module. The aggregate changes being made to the global network parameters by these distinct actor-learners performing asynchronous updates in parallel are therefore likely to be less correlated because they are using different exploration policies in different threads of the actor-learner module.

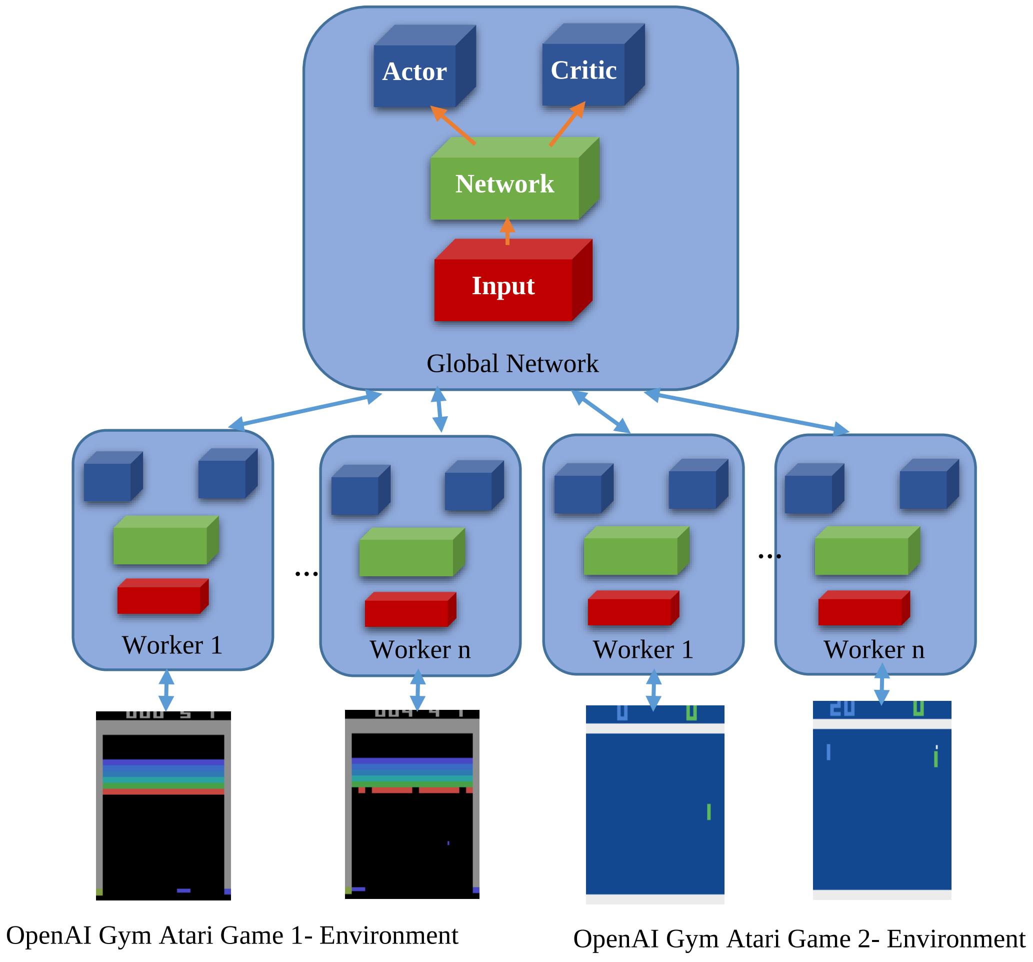


Figure 3.9. Architecture of Hybrid Parallel Multi-Task Model

3.9. Chapter Summary

In this chapter, the way to improve reinforcement learning from a multi-task learning perspective with A3C has been discussed. The original objective was to enhance the performance of DRL agents, so a hybrid multi-task learning model was created. The performance of the DRL agent is assessed using a hybrid multi-task learning model with partial observability, efficient exploration, and the quantity of training data and training time necessary to achieve the acceptable performance level.

Chapter 4

Implementation and Experimental Results

The objective of this system is to achieve optimal performance by leveraging different perspectives and strategies. Through the integration of distinct perspectives, the system aims to enhance the overall efficiency and effectiveness of deep reinforcement learning algorithms. In this section, the implementation details of the Hybrid Multi-Task Deep Reinforcement Learning System are presented, providing insights into the architectural design and algorithmic components. Following the implementation overview, the experimental results obtained from extensive testing and evaluation Through a series of rigorous experiments, the system performance across various benchmark tasks and domains are assessed. These experiments encompass a comprehensive analysis of the capabilities of the system, highlighting its strengths and areas for improvement.

4.1. System Design

This system utilized the prototype implementation of the hybrid multi-task paradigm in the A3C based algorithm. The prototype was evaluated during implementation using a variety of games in the setting of an Atari 2600 provided by the OpenAI Gym. OpenAI created the Gym library as a toolset for creating and contrasting RL algorithms. The A3C algorithm was applied to the game environment Breakout-v0 to create the first stage of the hybrid multi-task model. The actor-critic methodology is the core model of the high-level architecture.

The Breakout agent always starts with a paddle and a brick wall at the top of the screen and the goal is to strike the bricks with the ball in order to maximize reward while catching the ball with the paddle to preserve life. The newest state (4 screens) is entered into Actor, which is being carried out by the Actor-network with its current settings, and it outputs the action number for state. The Prediction performed by the Critic-network with its current parameters receives the most recent state (4 screens) as input and generates the anticipated reward for each state.

The environment (ALE, an Atari game emulator that is employed in this work), handles the Act stage. It uses the anticipated action as input and outputs the following game screen, the outcome reward, and whether or not the game ends. The Observation adding a new tuple (consisting of the predicted action, the reward of the action, the new state, the current state, and the terminal value) as an experience is done after t_{max}. Bootstraps are got from the last state of t_{max} that is predicted by critic network or the last state is terminal, bootstraps is zero. Learning from the experiences, calculates advantage, reward from the future states plus the actual reward of state t, actor loss and value loss. The agent updates the calculated parameters to the global network and then copies back the last parameters from the global network to the actor and critic networks. At iteration t the learning modifies the Actor network and Critic network by minimizing actor loss and value loss. At the end of the global step, the agent gets the best parameters from the global network.

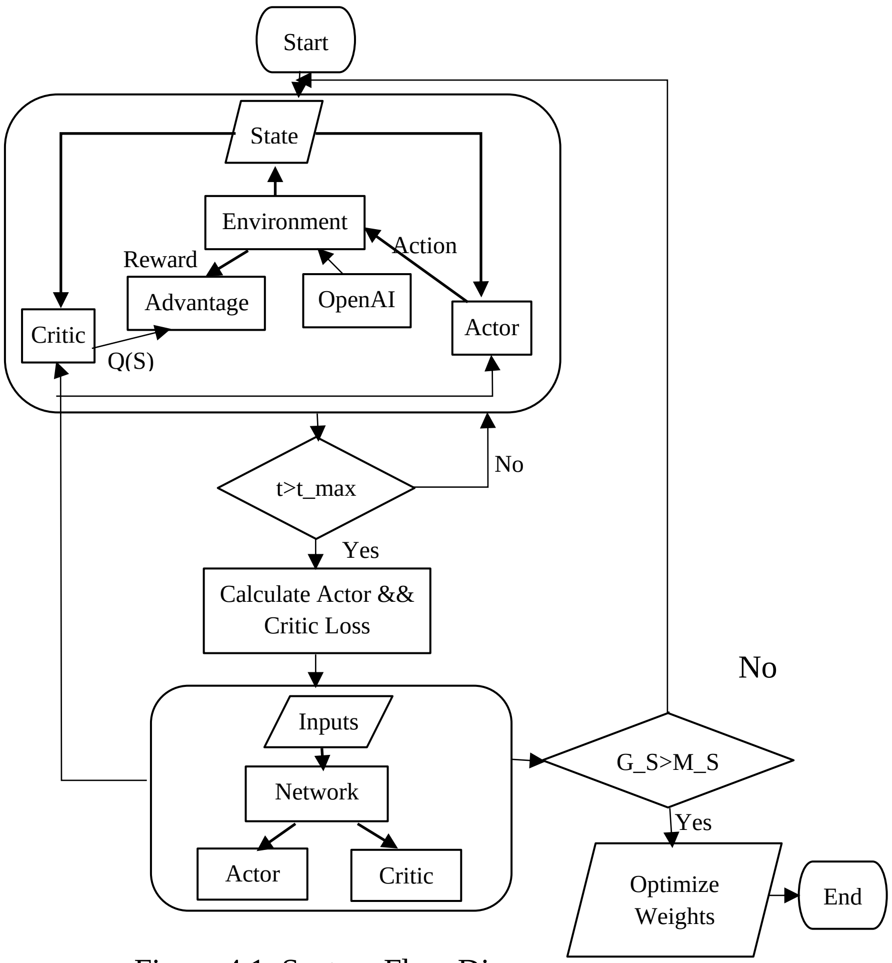


Figure 4.1. System Flow Diagram

4.2. Dataset Description

Breakout-v4, a more graphically advanced Atari 2600 gaming environment, is handled as a complicated environment since there are an infinite number of state action spaces to manage. Many of the games available in OpenAI Gym Atari are based on classic arcade games, which are simple enough to be understood and debugged easily, but complex enough to be challenging for AI algorithms. The neural network-based model was employed for the validation in order to accommodate and manage this environment. The state used for the Breakout game, or the previous four frames, is an illustration of a finite discrete state space. The state is an array of 4x210x160 integers since it is made up of 4x210x160 pictures concatenated channel-wise. A pixel can accept values between 0 and 255. As a result, there are $255^{4*210*160}$ potential states in the state space. Though enormous, the state space is finite.

Table 4.1. Global Step and Frames (Images)

Steps	Frames
1	4
1e6	4e6
1e7	4e7

Table 4.2. Description of Some Atari 2600 Games from Experiment

Game	Height	Width	Channel	Maximum Reward	Minimum Reward	Action Space
Breakout	210	160	3	864	0	4
Pong	210	160	3	21	-21	4
Space Invaders	210	160	3	9999(display) Note: More than 9999	0	6
Demon Attack	210	160	3	0	Unlimited	6
River Raid	210	160	3	0	Unlimited	18
Adventure	250	160	3	0	Win	18

Table 4.3. All Parameters for Neural Networks from Experiment

Parameter	**Amount**
Global Norm	40
Clip by Value	Min=1e-20, Max=1
Entropy Probability	0.01
Discount Factor	0.99
Learning Rate	0.00025
Decay	0.99
Momentum	0.0
Epsilon	1e-6

Table 4.3, wherein **Global Norm** is the normalization for global network that even with an irregular loss terrain, gradient descent can behave reasonably because of gradient clipping. Without clipping, the parameters leave the "excellent" region and descend rapidly. With clipping, the parameters remain in the "good" zone and the descent step size is constrained. Value Clip for policy network of workers, with its values trimmed to clip value min and clip value max, this procedure produces a tensor with the same type. Clip value min is set to any value that is less than that value. Clip value max is set to any values that are greater than it. $P=0.01$ for exploration of agent to get new knowledge and $(1\text{-}P)$ takes advantage of the present estimated worth of the agent and opts for the greedy strategy to reap the most rewards. The agent might not receive the highest reward since it is being greedy with the estimated value rather than the actual value.

The reinforcement learning level of concern about rewards in the distant future in comparison to those in the near future is primarily determined by the discount factor. The agent will only learn about acts that result in an immediate reward if $\gamma=0$, making it fully myopic. If $\gamma=1$, the agent will assess each of its decisions in light of the aggregate value of all of its potential rewards. In this experiment chooses $\gamma = 0.99$, the agent will evaluate its decision on 99% of the sum of total of all of its future rewards. A hyper-parameter called $\alpha = 0.00025$ is used to control how quickly an algorithm learns or updates the values of a parameter estimate. In other words, the neural network weights with respect to the loss gradient are controlled by the learning rate.

In RMSProp, a variation of gradient descent, the step size for each parameter is adjusted using a declining average of partial gradients. The technique can ignore early gradients and concentrate on the most recent partial gradients seen as the search progresses by using a decaying moving average (Decay=0.99, the best choice for the most of neural network). A straightforward method called neural network momentum frequently increases both training speed and precision. Neural Network weights worth could skyrocket. A parameter is included called epsilon (ε) with a small value in the denominator to stop the gradients from blowing up.

4.3. Experimental Results and System Analysis

The hardware and software specifications for this experiment include 8 GB of RAM, Tensor version 1, and a 64-bit operating system. The simulation was developed on a contemporary local computer, utilizing only the Python programming language and relying on CPU-based computations (Threads).

Processor	Intel(R) Core (TM) i5-1135G7 @ 2.40GHz 2.40 GHz
Installed RAM	8.00 GB
Window Edition	Window 11 Home
Operating System	64-bit operating system, x64-based processor
Python Version	3.7
Tensor Version	1.15
Training On	CPU (Threads)

Each individual worker agent in the A3C-based multi-task worker agent environment is directly governed by the global network. Using parameter values shared by the entire network, each worker is initially reset in this technique. Later, the worker interacts with its own copy of the environment. Each of the worker agents is initialized differently even though they are all working in the same game environment. As a result, everyone of these agents may begin at a different location in their environment. Each worker agent participates in a certain number of game episodes throughout operation and determines the value and corresponding policy loss. Since the neural network is used to build both the actor and critic modules, gradient values are derived from the losses while operating. After the worker agent has completed a predetermined number of game episodes, these gradient values will be broadcast to the entire network. The implementation of the simulation is done by using pure python with tensor.

In Table 4.4, the thesis focuses on hybrid multi-task environments aiming to enhance the performance of a global network using different network structures. Specifically, these network structures consist of only CNN layers or a combination of CNN and LSTM layers to incorporate long-term memory. The performance of these hybrid environments is evaluated based on two key factors: training time and average rewards. To conduct the study, two simulation games, namely Breakout and Pong environments, are chosen as the initial testbeds. Both games share the same minimal action numbers and exhibit similar objectives. Additionally, two other games, Space Invaders and Demon Attack, which involve a broader set of six action numbers.

By exploring the hybrid multi-task environments and their network structures, the research aims to analyze and optimize the performance of the global network. The evaluation criteria consider the training time required and the average rewards obtained throughout the learning process. The maximum rewards of CNN and LSTM layers are bigger than only CNN for four games such as Pong, Space Invaders, Demon Attack, and River Raid. Nevertheless, the average training time for CNN layers is less than CNN and LSTM layers in hybrid multi-tasking.

Table 4.4. Implementation of Hybrid A3C with Groups of Two Games

Game	Neural Network	Global Steps	Training Time	Batch Size	Worker	Maximum Rewards	Minimum Rewards
Breakout	CNN and LSTM	1e7	1 day	20	4	124	0
Pong		1e7			4	1	-21
Space Invaders	CNN and LSTM	1e7	13 hours	20	4	1070	10
Demon Attack		1e7			4	2640	0
River Raid	CNN and LSTM	1e7	18 hours	20	4	5510	180
Adventure		1e7			4	0	0
Breakout	CNN	1e7	20 hours	5	4	244	0
Pong		1e7			4	-5	-21
Space Invaders	CNN	1e7	13 hours	5	4	1020	5
Demon Attack		1e7			4	1760	0
River Raid	CNN	1e7	17 hours	5	4	3760	280
Adventure		1e7			4	0	0

In Table 4.5, the results of the initial training phase for the breakout game are presented. The table showcases data for 10 episodes, each with various variables including Reward, Global Step, Predict Total Reward, Actions Counts, Average Advantage, Average Value Loss, Average Policy Loss, and Steps per Episode. Notably, episodes 9 and 10 stand out as they achieved the highest reward. Furthermore, episode 10 recorded a maximum predict total reward of 23.80739, while episode 9 had the highest steps per count at 334.

Table 4.5. First 10 Episodes of Breakout Game before Training

Reward	Global Step	Predict Total Reward	Action Counts [0 1 2 3]	Average Advantage	Average Value Loss	Average Policy Loss	Steps per Episode
0	714	6.183332	[48 47 56 39]	-0.00092	0.000156	-0.08186	190
2	942	7.736671	[49 83 60 55]	0.023142	0.117418	0.089122	247
2	1089	7.560577	[63 72 61 65]	0.029204	0.146041	0.13072	261
2	1152	7.585649	[69 64 72 70]	0.017259	0.089341	0.049834	275
0	1356	2.194534	[51 38 38 46]	-0.0004	6.84E-06	-0.07049	173
2	2056	-2.88062	[64 77 81 71]	0.03051	0.147595	0.139584	293
1	2275	2.576762	[79 48 50 70]	0.011722	0.060055	0.003141	247
2	2458	6.429946	[90 67 81 74]	0.012183	0.063653	0.006725	312
3	2562	9.11074	[93 86 89 66]	0.017096	0.088782	0.047676	334
3	3267	23.80739	[77 84 77 92]	0.027903	0.149046	0.109676	330

The experimental results for the pong game are presented in Table 4.6, showcasing the initial training phase consisting of 10 episodes. Throughout the training process, the average advantage and average policy loss consistently maintained negative values, indicating a significant learning curve. On the other hand, the average value loss displayed a different trend, suggesting potential improvements in this aspect.

Table 4.6. First 10 Episodes of Pong Game before Training

Reward	Global Step	Predict Total Reward	Action Counts [0 1 2 3]	Average Advantage	Average Value Loss	Average Policy Loss	Steps per episode
-21	4526	-214.192	[272 264 256 288]	-0.04005	0.234107	-0.35986	1080
-20	4733	-243.309	[284 311 307 330]	-0.04538	0.274845	-0.38106	1232
-21	4806	-258.116	[314 285 338 341]	-0.03895	0.223264	-0.34872	1278
-19	5550	-276.458	[308 333 324 331]	-0.03555	0.235196	-0.30104	1296
-21	9000	-446.026	[366 177 249 369]	-0.04795	0.285827	-0.39112	1161
-21	9157	-386.675	[314 194 199 341]	-0.03886	0.251239	-0.33609	1048
-20	9529	-477.333	[387 209 281 428]	-0.03671	0.24232	-0.31057	1305
-21	10898	-501.937	[374 198 255 386]	-0.03173	0.218115	-0.28483	1213
-20	13537	-548.673	[316 240 320 350]	-0.03573	0.274691	-0.30791	1226
-21	14112	-562.859	[333 243 304 343]	-0.02994	0.212771	-0.28099	1223

Table 4.7. (10) Episodes after Training 10 million Steps of Breakout Game

Reward	Global Step	Predict Total Reward	Action Counts [0 1 2 3]	Average Advantage	Average Value Loss	Average Policy Loss	Steps per Episode
50	6316	3410.365	[429 446 297 362]	0.056979	4.901371	0.527547	1534
42	6894	3895.781	[493 547 328 420]	0.03546	3.739859	0.459458	1788
67	7719	4295.355	[483 629 341 426]	0.114926	6.712981	1.818685	1879
72	7780	4490.385	[487 639 374 489]	0.135794	6.810067	2.539582	1989
47	12280	3282.468	[307 525 236 301]	0.118526	7.515264	2.079136	1369
41	13001	2984.526	[281 509 225 265]	0.133405	7.934946	2.116966	1280
72	14134	4641.899	[466 737 323 399]	0.131398	7.072025	2.299549	1925
66	15646	4724.153	[467 783 328 419]	0.116403	7.44021	1.904273	1997
51	18687	3858.689	[393 583 310 352]	0.088347	6.396375	1.19495	1638
90	20870	4877.574	[506 641 336 444]	0.225278	12.05578	4.290097	1927

Table 4.7 showcases the results of 10 episodes following 10 million steps of training in the breakout game. Notably, all episodes achieved a reward exceeding 40, which represents significant progress compared to the initial training phase where the maximum reward reached only 3. This demonstrates the substantial improvement in the agent's performance as a result of the extensive training.

Furthermore, the predict total reward after 10 million steps (with a minimum of 2984) demonstrates a notable divergence from the initial training phase (with a minimum of -2.88062). This substantial increase highlights the agent's enhanced ability to predict and accumulate rewards over time.

However, it is worth noting that the average advantage, average value loss, and average policy loss in these 10 episodes are larger compared to the initial training phase. While this may indicate some challenges during the later stages of training, it also suggests that the agent is exploring new strategies and potentially encountering more complex scenarios. Overall, these findings underscore the remarkable progress made by the agent in the breakout game after 10 million steps of training, particularly in terms of rewards and predictability, despite certain areas requiring further refinement.

An interesting observation is the high number of steps per episode, exceeding 1,000 steps for each episode. This indicates that the agent was actively engaged in the game, demonstrating its ability to handle complex gameplay scenarios. Overall, these findings highlight the promising progress made during the early stages of training for the pong game.

Table 4.8. (10) Episodes after Training 10 million Steps of Pong Game

Reward	Global Step	Predict Total Reward	Action Counts [0 1 2 3]	Average Advantage	Average Value Loss	Average Policy Loss	Steps per Episode
-15	22700	-503.648	[1412 1511 1411 1472]	-0.01341	0.764262	-0.59473	5806
-8	24030	-466.635	[1480 1462 1419 1484]	-0.00052	1.031243	-0.27583	5845
-13	24171	-530.477	[1509 1607 1518 1563]	-0.01419	0.8474	-0.63709	6197
-11	27617	-571.503	[1707 1702 1630 1718]	-0.00507	0.722031	-0.43404	6757
-20	29737	-942.781	[388 405 501 517]	-0.06441	2.072405	-1.57704	1811
-13	35624	-1160.62	[659 703 770 787]	-0.00031	1.099874	-0.34307	2919
-16	40326	-1376.25	[919 959 971 1087]	-0.00258	0.763664	-0.27548	3936
-17	40647	-1285.04	[712 766 806 907]	-0.00693	0.936628	-0.36593	3191
-14	54184	-1289.51	[1594 1588 1576 1626]	-0.00855	0.718837	-0.51759	6384
-15	54906	-602.091	[865 852 859 918]	-0.02443	1.116961	-0.8899	3494

Table 4.8 presents the results of the pong game training, specifically highlighting the reward of second episode is -8. Additionally, the maximum predict total reward achieved after 10 million steps is recorded as -466.635. These results showcase the challenges encountered during the training process, as the agent struggled to achieve positive rewards consistently.

Furthermore, a noteworthy observation is that the average advantage significantly decreased compared with the beginning in training. This suggests that, as the training progressed, the agent faced difficulties in gaining a competitive advantage or consistently outperforming its previous performance.

These findings underscore the complexity of the pong game and the ongoing learning process for the agent. Despite encountering challenges and obtaining suboptimal rewards, the training data provides valuable insights for further improvements and adjustments in the agent's training methodology.

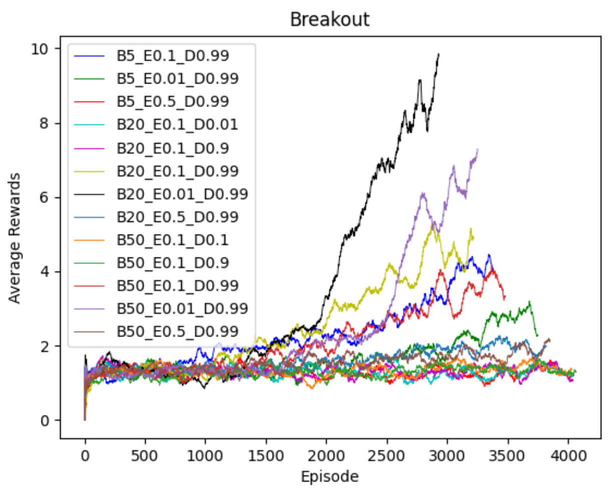


Figure 4.2. Comparison between Different Hyperparameters Values

In Figure 4.2, where *B* is the batch size, or maximum time step t_{max}, *E* is the entropy value and *D* is the discount factor. Only one million steps can make a greedy converge on the most of various hyperparameters for breakout game. The highest-ranking black line is batch size = 20, entropy = 0, and discount factor = 0.99. The second line is batch size = 50, and the entropy and discount factor are the same as the highest line.

Table 4.9. Training Time Comparison between Different Hyperparameters Values

Entropy	Discount	Batch Size	Training Time
0.01	0.99	20	1:16:25.699166
0.1	0.99	20	1:02:20.585934
0.5	0.99	20	1:02:33.721475
0.01	0.99	5	1:24:10.260708
0.1	0.99	5	1:49:45.386151
0.5	0.99	5	1:24:57.743445
0.01	0.99	50	0:59:42.703493
0.1	0.99	50	0:51:30.136267
0.5	0.99	50	0:46:29.638318
0.1	0.9	50	1:00:59.892897
0.1	0.1	50	0:43:43.854129
0.1	0.1	20	1:06:33.276819
0.1	0.9	20	0:52:17.657674

The experimentation involved three different entropy values, *E= {0.01, 0.1, 0.5}*, three discount factor values, *D= {0.1, 0.9, 0.99}*, and three batch sizes, *B= {5, 20, 50}*. These experiments exclusively focused on breakout games, and it was observed that using a batch size of 50 resulted in faster overall training compared to batch sizes of 20 and 5.

Notably, the combination of *E=0.1*, *D=0.1*, and *B=50* exhibited the shortest training time among all the experiments conducted. This particular configuration allowed for efficient training with optimal results. On the other hand, the combination of *E=0.5*, *D=0.99*, and *B=50* secured the second-best performance, showcasing significant progress and demonstrating the effectiveness of these parameters. Overall, these findings highlight the impact of different entropy, discount factor, and batch size values on the training process in breakout games. They provide valuable insights into optimizing training time and achieving notable performance gains.

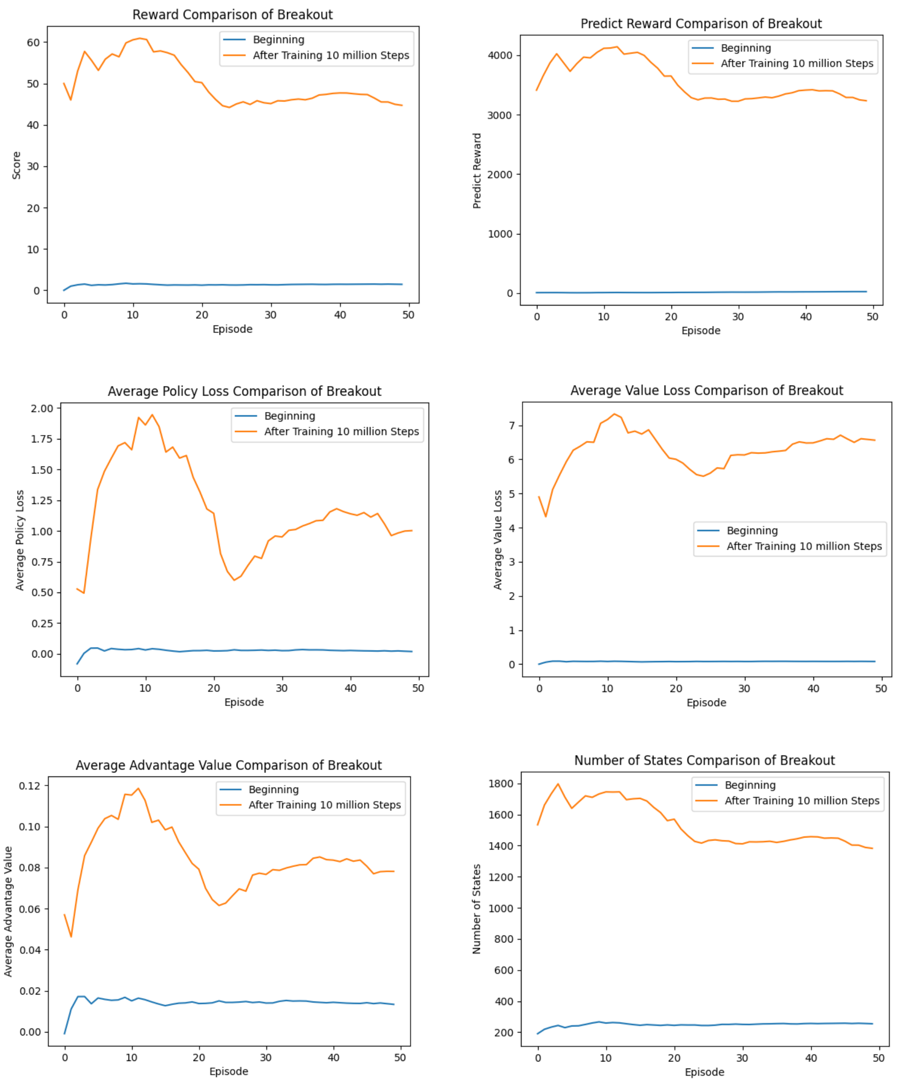


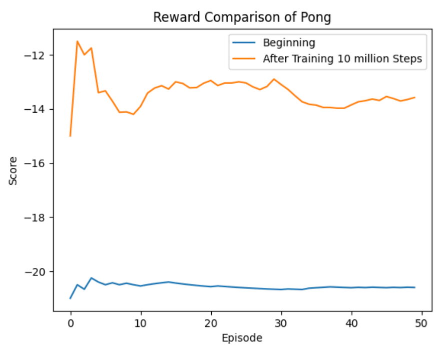
Figure 4.3. Test Results of 50 Episodes of Breakout Game

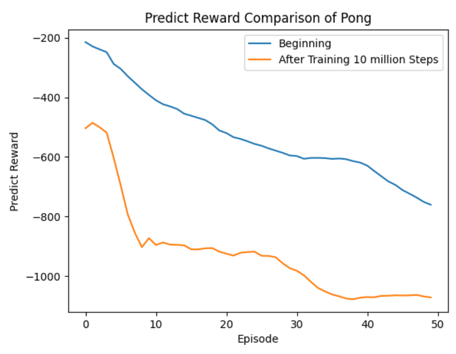
In Figure 4.3, several statistics are presented for the Breakout game. The first image shows the comparison between the initial training rewards and the rewards obtained after the final training, based on 50 episodes. The second image represents the predicted total rewards derived from all states using the value network.

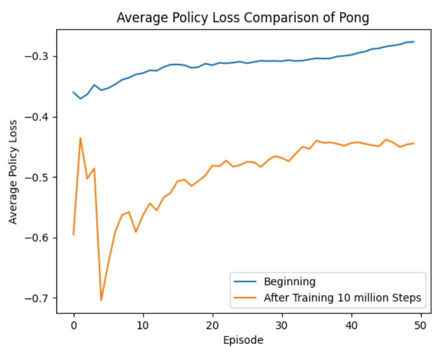
The statistic also provides information about the average policy loss, which is calculated by summing the policy losses of 20 steps (batch size). Similarly, the average value loss is calculated by summing the value losses of 20 steps. Furthermore, the average advantage function is computed at the difference between the actual rewards and the predicted rewards.

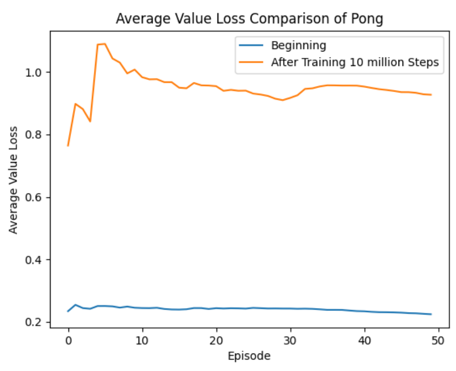
Additionally, the result highlights the total number of states encountered by the agent in each episode. By analyzing these statistics, a significant improvement is observed after training the agent for 10 million steps with four workers. The total rewards achieved by the agent consistently surpass an average of 40 rewards per episode, even after a relatively short training period. Moreover, the agent demonstrates the ability to predict which state s_t leads to better long-term rewards.

The policy gradient-based actor employed successfully increases the rewards throughout the episodes. The value-based critic module also shows an improved performance, particularly in the early episodes. By comparing the actual rewards with the expected rewards, the agent gains insights into the optimal policy for each state. Consequently, the agent can effectively explore a large number of states in its environment, albeit at a later stage.









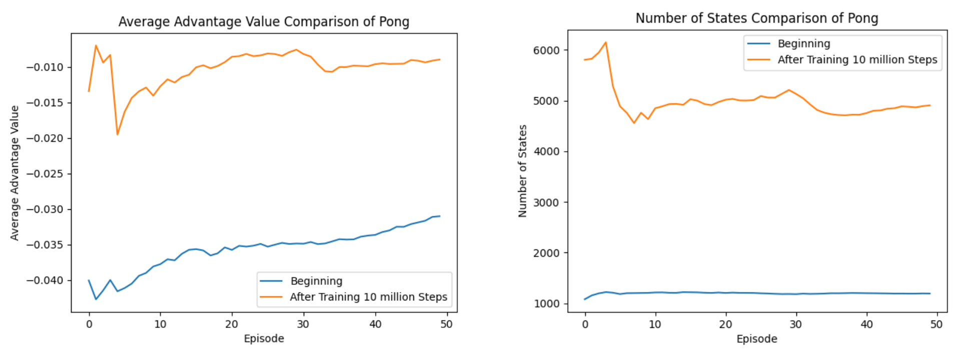


Figure 4.4. Test Results of 50 Episodes of Pong Game

The initial image shows the results of 50 episodes of the Pong game, displaying the training rewards before and after the final training phase. In each episode, the second image represents the predicted total rewards obtained from all states using the value network. Additionally, the average policy loss is calculated by summing the policy losses of 20 steps (batch size) and the average value loss is computed by summing the value losses of 20 steps as well. Moreover, the average advantage function, which quantifies the difference between actual rewards and predicted rewards, is also included. Lastly, the total number of states encountered by the agent in each episode is recorded.

From these statistics, a significant disparity is observed between the performance before and after training for 10 million steps with 4 workers in the Pong game. Despite the game being partially observable and stochastic, the total rewards consistently surpass the average of -13 rewards per episode after a brief training period. Moreover, the predictions indicate an increasing trend in rewards for all states over an extended training duration.

The actor component effectively manages to minimize the loss associated with each action, gradually approaching the global minima. As the training progresses, the value-based critic module anticipates the improvement of performance for each agent. Notably, the agent demonstrates steady and accurate determination of the advantage function over time. Additionally, the agent encounters nearly six times the number of environmental states throughout the games.

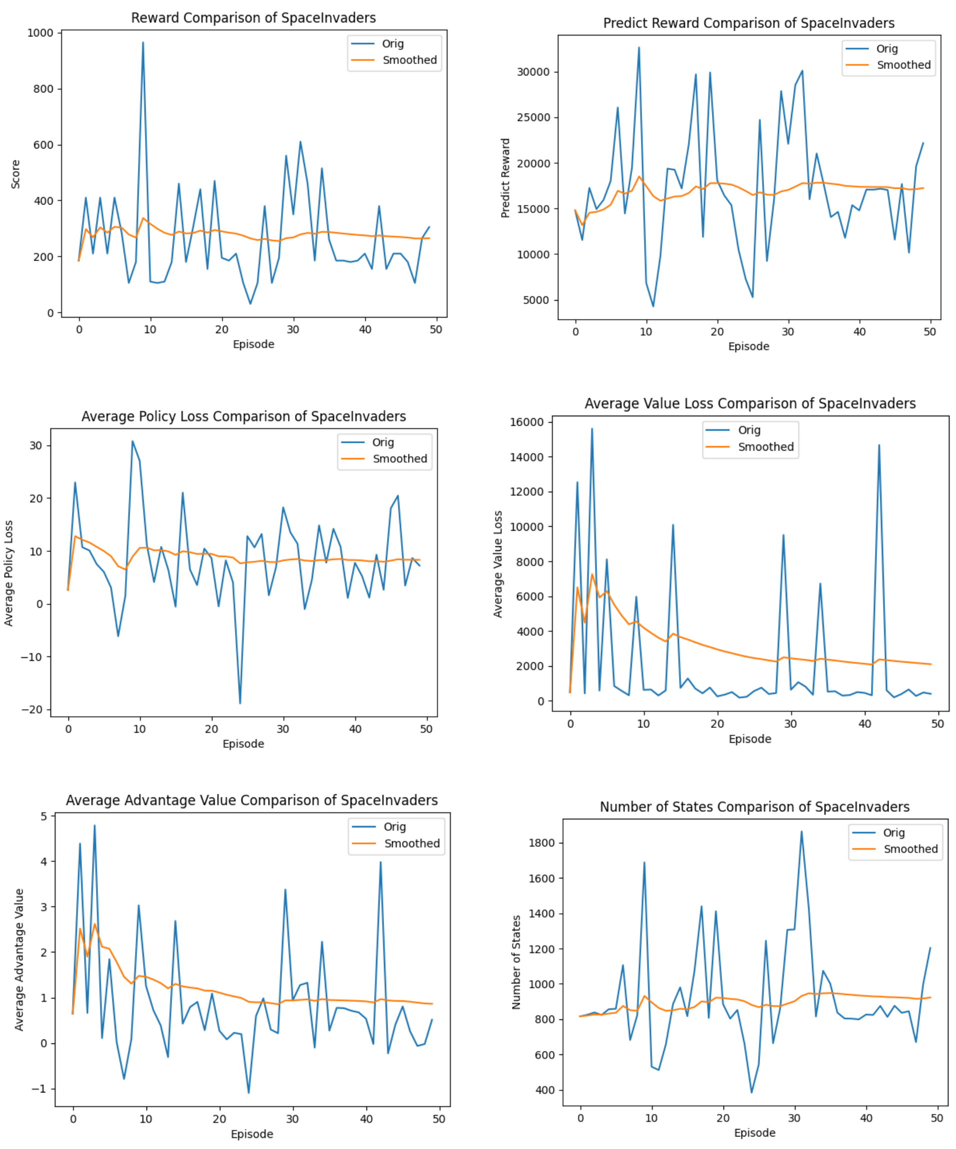


Figure 4.5. Test Results of 50 Episodes of SpaceInvaders Game

The initial image displays the outcomes after the final training phase of 50 episodes in the Space Invaders game, with the average rewards represented by the orange line. The second image showcases the predicted total rewards obtained from all states using the value network. Subsequently, the average policy loss is calculated by summing the policy losses of 20 steps (batch size), while the average value loss is computed by summing the value losses of 20 steps as well. Additionally, the average advantage function, which captures the discrepancy between actual rewards and predicted rewards, is included. Lastly, the total number of states encountered by the agent in each episode is recorded.

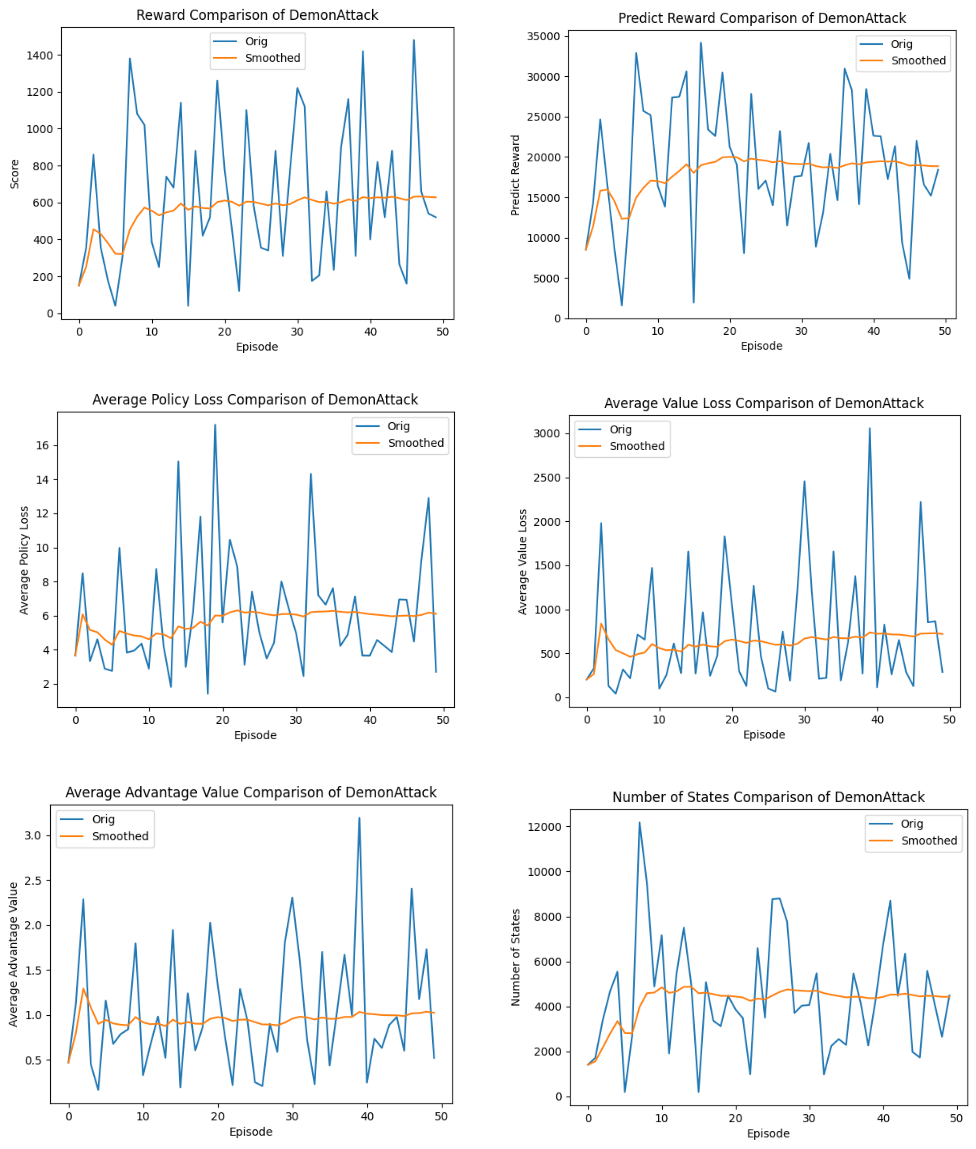


Figure 4.6. Test Results of 50 Episodes of DemonAttack Game

The first image exhibits the results obtained after the final training of 50 episodes in the Demon Attack game. Following that, the second image presents the predicted total rewards derived from all states through the value network. Moving forward, the average policy loss, calculated by summing the policy losses of 20 steps (batch size), is shown, along with the average value loss, which is also computed using the sum of 20 steps (batch size). Furthermore, the average advantage function, representing the difference between the actual rewards and the predicted rewards, is depicted. Lastly, the total number of states faced by the agent in each episode is recorded.

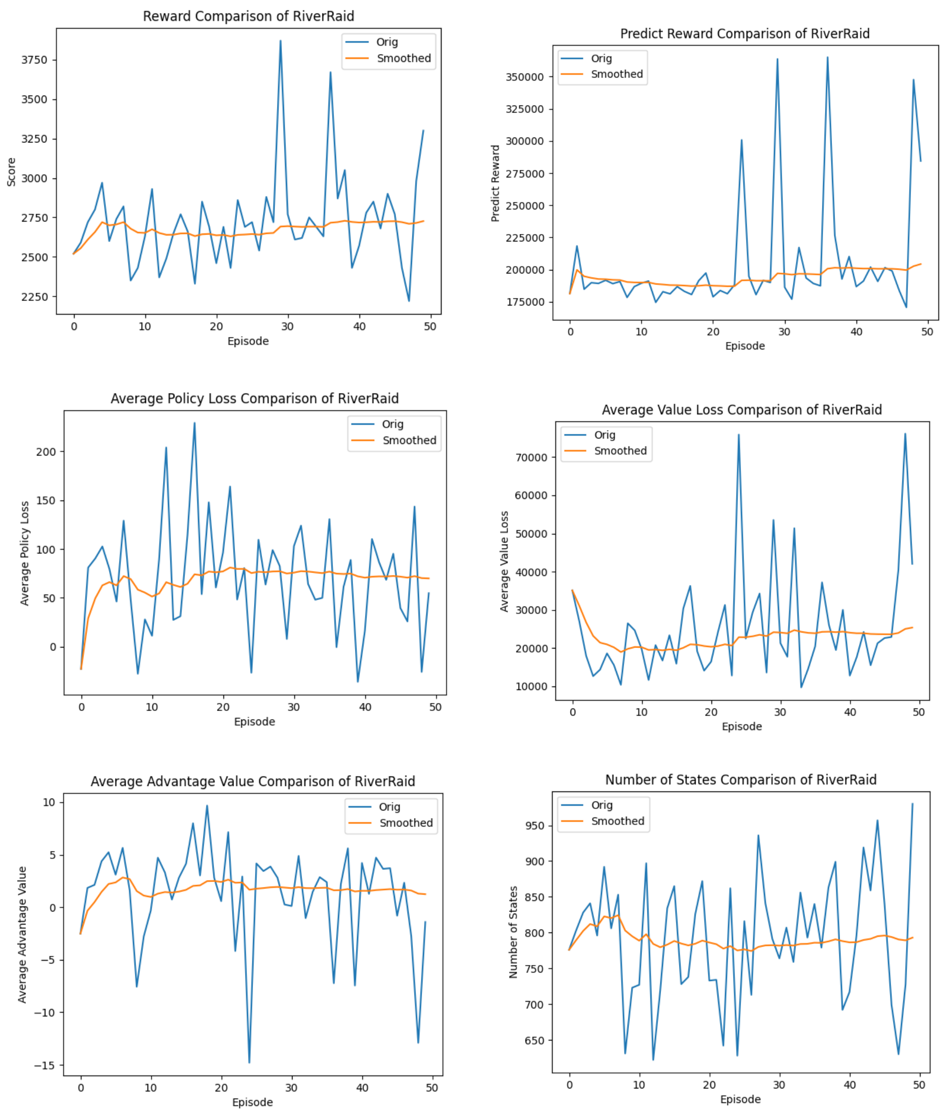


Figure 4.7. Test Results of 50 Episodes of RiverRaid Game

The first image displays the outcomes after the final training of 50 episodes in the River Raid game. It represents the performance achieved by the agent following extensive training. The second image depicts the predicted total rewards obtained from all states using the value network. This provides an insight into the expected cumulative rewards for agents. Additionally, the average policy loss, calculated at the sum of policy losses over 20 steps (batch size), is shown. Similarly, the average value loss, obtained by summing the value losses over 20 steps (batch size), is presented. These metrics highlight the training progress and convergence of the agent policies and value estimation. Furthermore, the average advantage function, which quantifies the difference between actual rewards and predicted rewards, is included. It reflects the agent ability to assess the value of its actions accurately. Lastly, the total number of states encountered by the agent in each episode is recorded, illustrating the complexity and variability of the game environment.

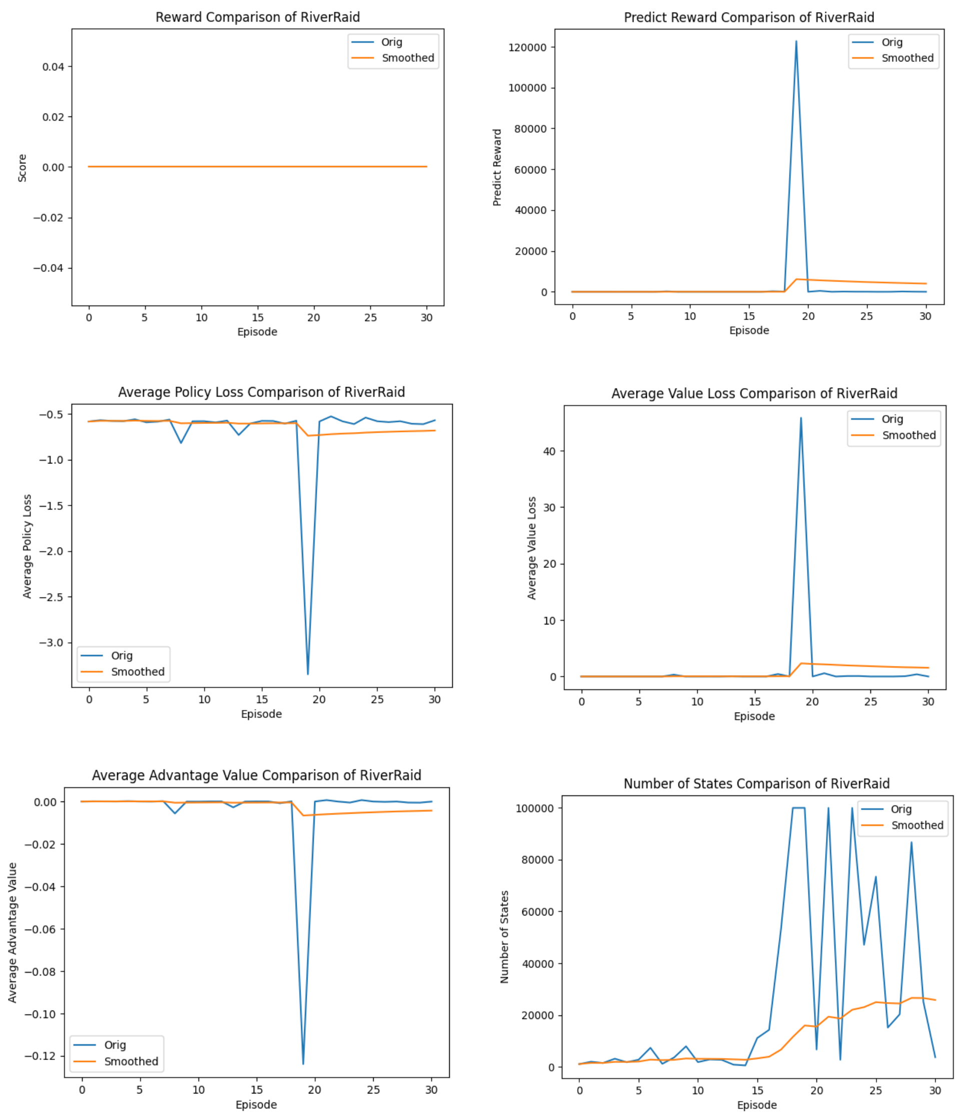


Figure 4.8. Test Results of 50 Episodes of Adventure Game

The first image showcases the outcomes obtained after the final training of 50 episodes in the Adventure game. It represents the performance of an agent after extensive training. The second image portrays the predicted total rewards derived from all states through the value network, offering insight into the expected cumulative rewards of agents. Moving forward, the average policy loss is presented, which is calculated by summing the policy losses over 20 steps (batch size). Similarly, the average value loss, obtained by summing the value losses over 20 steps (batch size), is displayed. These metrics provide an understanding of the training progress and convergence of agent policies and value estimation. Additionally, the average advantage function is included, quantifying the discrepancy between the actual rewards and the predicted rewards. This metric demonstrates the ability of an agent to accurately assess the advantage of its actions. Finally, the total number of states encountered by the agent in each episode is recorded, highlighting the complexity and variety of the game environment.

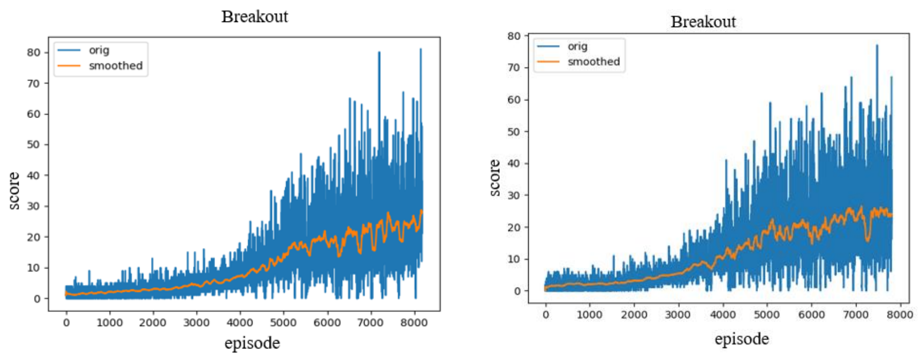


Figure 4.9. Comparsion between 8 Workers and 4 Workers of Training Time and Performance (Rewards and Average Rewards) with Breakout Game

In Figure 4.9, both are Breakout games comparing 4 workers and 8 workers of training time and the performance with CNN network. The first picture shows 8 workers and training time is 5:30 hours in 5 million global steps. The second picture shows 4 workers and training time is 6:45 hours in 5 million global steps.

4.4. Discussion on Experimental Results

This experiment is implemented on six Atari 2600 games supported by OpenAI using two neural networks and deep learning. The amount of training steps required to get a certain score for a given task and algorithm should, in an ideal scenario, remain constant with varying numbers of workers when using many workers concurrently and updating a shared model. The advantage of the system lies in its ability to process a larger volume of data within the same timeframe, potentially leading to increased exploration. Figure 4.9 demonstrates that the effective using of resources and growing effectively with the number of simultaneous workers.

According to the test results, the scores of the individual games were significantly impacted by either negative information transfer or the gradients shared between workers training threads with different semantic properties. The test results show that positive knowledge transfer significantly increased the performance of each individual game. Bigger batch size and adding LSTM give better results than only with CNN layers although training time as the same in both networks. Only if Adventure game, others predict more rewards than the beginning of life from the critic networks. All of six games developed their intelligence such Advantage function went nearly zero that mean they could almost correctly predict the value of the states after training. According to the line graphs and due to the impact of positive knowledges, A3C makes efficiency in losing of actor network weights and losing of critic network weights for two similar multi-task environments.

4.5. Chapter Summary

This chapter encompasses the system implementation and experiments conducted for hybrid multi-task deep reinforcement learning. The chapter initiates by presenting the experimental configuration employed for the system. Subsequently, the experimental results obtained from the system are meticulously analyzed. To provide a comprehensive understanding of the experimental outcomes, this chapter focuses on the scores and training times of different network structures, different environments, and different hyperparameters. Moreover, a detailed comparison of all the results is presented, followed by the conclusions derived from the system evaluation.

Chapter 5

Conclusion and Future Works

5.1. Conclusion

The hybrid multi-task A3C (Asynchronous Advantage Actor-Critic) algorithm offers a remarkable capability to learn directly from raw pixel inputs, eliminating the need for manually engineered features. This attribute makes it particularly well-suited for domains characterized by complex sensory inputs. By introducing asynchronous parallelism, A3C overcomes the limitations of traditional reinforcement learning methods, enabling multiple agents to explore the environment concurrently and learn independently. The utilization of parallelization in A3C brings significant benefits, including accelerated training and improved sample efficiency. This approach leverages modern hardware architectures, such as multi-core CPUs or distributed computing setups, effectively harnessing their computational power.

In the context of five Atari 2600 games, employing a batch size of 20, along with appropriate normalization for both the global network and worker networks, has proven to be more efficient than a batch size of 5. The combination of network layers, namely Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM), in the hybrid multi-task A3C framework consistently outperforms using only CNN layers, as indicated by the average rewards achieved by four workers in each game.

The Hybrid Multi-Task Deep Reinforcement Learning System, optimized from distinct perspectives, represents a significant advancement in the field of deep reinforcement learning. By leveraging diverse architectural designs and algorithmic components, the system showcases remarkable capabilities in optimizing training time, achieving higher average rewards, improving convergence rates, and exhibiting strong generalization abilities. Breakout - v4, Pong - v4, Demon Attack - v4, Space Invaders - v4, Adventure -v4, River Raid - v4 are used for the evaluation of this thesis. According to the viewpoint, this environment is being viewed as a potential field of a DRL agent. One that is difficult because there are an endless number of state action spaces to consider. The validation was performed using a model built on a convolutional neural network (CNN) and after the last hidden layer, a recurrent agent with 256 extra LSTM cells is used. This configuration was evaluated utilizing a multitask environment with four worker threads that together completed 10,000,000 game steps in a desktop-based environment. Although each thread has its own copy of the environment, they are all unique in how they perceive the gaming environment.

A3C algorithm-based multi-worker agent environment and hybrid multi-task model are also developed for Pong-v4, another graphically demanding Atari 2600 gaming environment. Pong-v4 was selected following a rigorous examination because Pong-v4 and Breakout-v4 have a lot in common with one another. An accelerator could be used to speed up the validation of the suggested hybrid multi-task learning model execution if there is a sufficient degree of similarity. The Pong-v4 game and Breakout-v4 were tested in a desktop-based environment using a hybrid multi-task environment with four worker threads for each game that collectively performed 10 million game steps.

The experiment is chosen one more set of Atari 2600 games from the Gym library, Space Invaders-v4 and DemonAttack-v4, to test hybrid multi-task paradigm. After considering the extensive semantic similarity between these two games' pattern play, it was decided to use them as the second test pair. Both of these games are based on the shooting genre and require the player to maneuver a moving ship that can fire and hit the adversaries. The gameplay in Demon Attack is more complicated from the perspective of the real-world agent because there are many different opponents who move more randomly and have the ability to shoot back.

More crucially, the Gym library provides built-in incentive structures for each game utilized in this experiment. In other words, even while the games selected for each test pair share certain semantic similarities, the scoring and reward systems used in each game are distinct. Whereas the adversaries in Space Invaders move more often than in the other game, making it comparably less complex in terms of intricacy. The SpaceInvaders-v4 game and DemonAttack-v4 were also tested using a desktop-based test environment, employing a hybrid multi-task worker model with four worker threads for each game that collectively ran around 10,000,000 game steps. This is similar to how the prior two games from the first pair were tested.

The last pair of this experiment is too difference in behaviour, these are Adventure-v4 and RiverRaid-v4. Adventure tries to solve the mystery of an evil magician while agent cannot get any reward on the other word agent cannot distinguish bad state or good. It is really the big deal of AI problem; the agent might be exhausted from this stochastic environment. RiverRaid-v4 is not quite different with SpaceInvaders-v4 and DemonAttack-v4 and only cares about how to get a lot rewards. Employing a hybrid multi-task worker model with four worker threads for each game that collectively ran around 10,000,000 game steps.

Updates exchanged across the global network, as opposed to a single game-based environment, should minimize some of the main issues related to partial observability because of their semantic similarity. The previous test results for each of the sets showed that each game inside each set might improve its performance throughout the training. The hybrid method to multi-tasking evaluates the impact of the amount of beneficial information transfer accomplished through parameter sharing. In comparison to a model-based approach, the chosen methodology is substantially less computationally intensive.

5.2. Limitation and Future Works

It is worth noting that the performance of the hybrid multi-task A3C can be sensitive to hyperparameter settings, necessitating extensive experimentation and tuning to discover optimal configurations. Additionally, the training process of hybrid multi-task A3C can be computationally demanding, especially when tackling complex tasks and operating within large-scale environments. One crucial observation from the experiments is that the effectiveness of the hybrid multi-task A3C heavily relies on the reward system. In situations where the environment fails to provide any feedback, the agent can become trapped in the unknown, resulting in difficulties in learning and progressing (e.g., in an adventure game).

As a future work, there is a potential for investigating the steps involved in navigating dissimilarity across multi-environments to aid in the development of intelligent agents while avoiding negative transfer. One direction to explore is extending the research to conduct hybrid multi-task experiments specifically in complex 3D environments and real-world scenarios, utilizing a single machine such as a microcomputer. It allows for a comprehensive evaluation of the performance of agents and adaptability in challenging and diverse environments, while also considering practical aspects such as scalability and resource utilization. By focusing on these future research directions, RL is contributed to advancing the field of intelligent agent development, paving the way for more robust and adaptable agents that can thrive in dissimilar multi-environments. In future work, there is potential for exploring the combination of hybrid multi-task A3C (Asynchronous Advantage Actor-Critic) with multi-agent systems. This entails extending the research to incorporate multiple agents (Multi-Agent System) working collaboratively or competitively within the same environment.

References

[1] V. Mnih et al., “Asynchronous methods for deep reinforcement learning.” arXiv, Jun. 16, 2016. Accessed: Jan. 14, 2023. [Online]. Available: http://arxiv.org/abs/1602.01783.

[2] V. Mnih et al., “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.

[3] D. Silver et al., “Mastering the game of Go with deep neural networks and tree search,” Nature, vol. 529, no. 7587, pp. 484–489, Jan. 2016, doi:10.1038/nature16961.

[4] S. Russell and Peter Norvig, “Artificial intelligence a modern approach,” Second. U.S, 2003.

[5] G. Weiss, Ed., “Multiagent systems: a modern approach to distributed artificial intelligence.” Cambridge, Mass: MIT Press, 1999.

[6] Y. Bengio, Ed., “Learning deep architectures for AI. in foundations and trends in machine learning,” no. Vol. 2, No. 1, 2009. Boston, Mass.: Now, 2009.

[7] R. S. Sutton and A. G. Barto, “Reinforcement learning: an introduction,” Second edition. in Adaptive computation and machine learning series. Cambridge, Massachusetts: The MIT Press, 2018.

[8] O. Sigaud, Ed., “Markov decision processes in artificial intelligence: MDPs, beyond MDPs and applications.” London: ISTE, 2010.

[9] D. Yarats, A. Zhang, I. Kostrikov, B. Amos, J. Pineau, and R. Fergus, “Improving sample efficiency in model-free reinforcement learning from images,” 2019, doi: 10.48550/ARXIV.1910.01741.

[10] Y. W. Teh et al., “Distral: robust multitask reinforcement learning.” arXiv, Jul. 13, 2017. Accessed: Jan. 14, 2023. [Online]. Available: http://arxiv.org/abs/1707.04175.

[11] S. J. Pan and Q. Yang, “A survey on transfer learning,” IEEE Trans. Knowl. Data Eng., vol. 22, no. 10, pp. 1345–1359, Oct. 2010, doi: 10.1109/TKDE.2009.191.

[12] N. Vithayathil Varghese and Q. H. Mahmoud, “A survey of multi-task deep reinforcement learning,” Electronics, vol. 9, no. 9, p. 1363, Aug. 2020, doi: 10.3390/electronics9091363.

[13] K. Weiss, T. M. Khoshgoftaar, and D. Wang, “A survey of transfer learning,” J. Big Data, vol. 3, no. 1, p. 9, Dec. 2016, doi: 10.1186/s40537-016-0043-6.

[14] A. A. Rusu *et al.*, “Progressive neural networks.” arXiv, Oct. 22, 2022. Accessed: Jan. 15, 2023. [Online]. Available: http://arxiv.org/abs/1606.04671.

[15] L. Espeholt et al., “IMPALA: Scalable distributed Deep-RL with importance weighted actor-learner architectures.” arXiv, Jun. 28, 2018. Accessed: Jan. 15, 2023. [Online]. Available: http://arxiv.org/abs/1802.01561.

[16] A. A. Rusu et al., “Policy distillation.” arXiv, Jan. 07, 2016. Accessed: Jan. 15, 2023. [Online]. Available: http://arxiv.org/abs/1511.06295.

[17] A. Mujika, “Multi-task learning with deep model based reinforcement learning.” arXiv, May 23, 2017. Accessed: Jan. 15, 2023. [Online]. Available: http://arxiv.org /abs/1611.01457.

[18] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska, “A survey of actor-critic reinforcement learning: standard and natural policy gradients,” IEEE Trans. Syst. Man Cybern. Part C Appl. Rev., vol. 42, no. 6, pp. 1291–1307, Nov. 2012, doi: 10.1109/TSMCC.2012.2218595.

[19] G. Brockman et al., “OpenAI gym.” arXiv, Jun. 05, 2016. Accessed: Jan. 15, 2023. [Online]. Available: http://arxiv.org/abs/1606.01540.

[20] A. Ghosh, A. Sufian, F. Sultana, A. Chakrabarti, and D. De, “Fundamental concepts of convolutional neural network,” vol. 172. Cham: Springer International Publishing, 2020, pp. 519–567. doi: 10.1007/978-3-030-32644-9_36.

[21] K. O’Shea and R. Nash, “An Introduction to convolutional neural networks.” arXiv, Dec. 02, 2015. Accessed: Jan. 15, 2023. [Online]. Available: http://arxiv.org /abs/1511.08458.

[22] A. Sherstinsky, “Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network,” Phys. Nonlinear Phenom., vol. 404, p. 132306, Mar. 2020, doi: 10.1016/j.physd.2019.132306.

[23] N. V. Varghese and Q. H. Mahmoud, “A hybrid multi-task learning approach for optimizing deep reinforcement learning agents,” IEEE Access, vol. 9, pp. 44681–44703, 2021, doi: 10.1109/ACCESS.2021.3065710.

[24] Z. Gu, Z. Jia, and H. Choset, “Adversary A3C for robust reinforcement learning.” arXiv, Dec. 01, 2019. Accessed: Jan. 15, 2023. [Online]. Available: http://arxiv.org /abs/1912.00330.

[25] N. V. Varghese and Q. H. Mahmoud, “Optimization of deep reinforcement learning with hybrid multi-task learning,” in 2021 IEEE International Systems Conference (SysCon), Vancouver, BC, Canada: IEEE, Apr. 2021, pp. 1–8. doi: 10.1109/SysCon48628.2021.9447080.

Appendix A.

Experimental Calculation

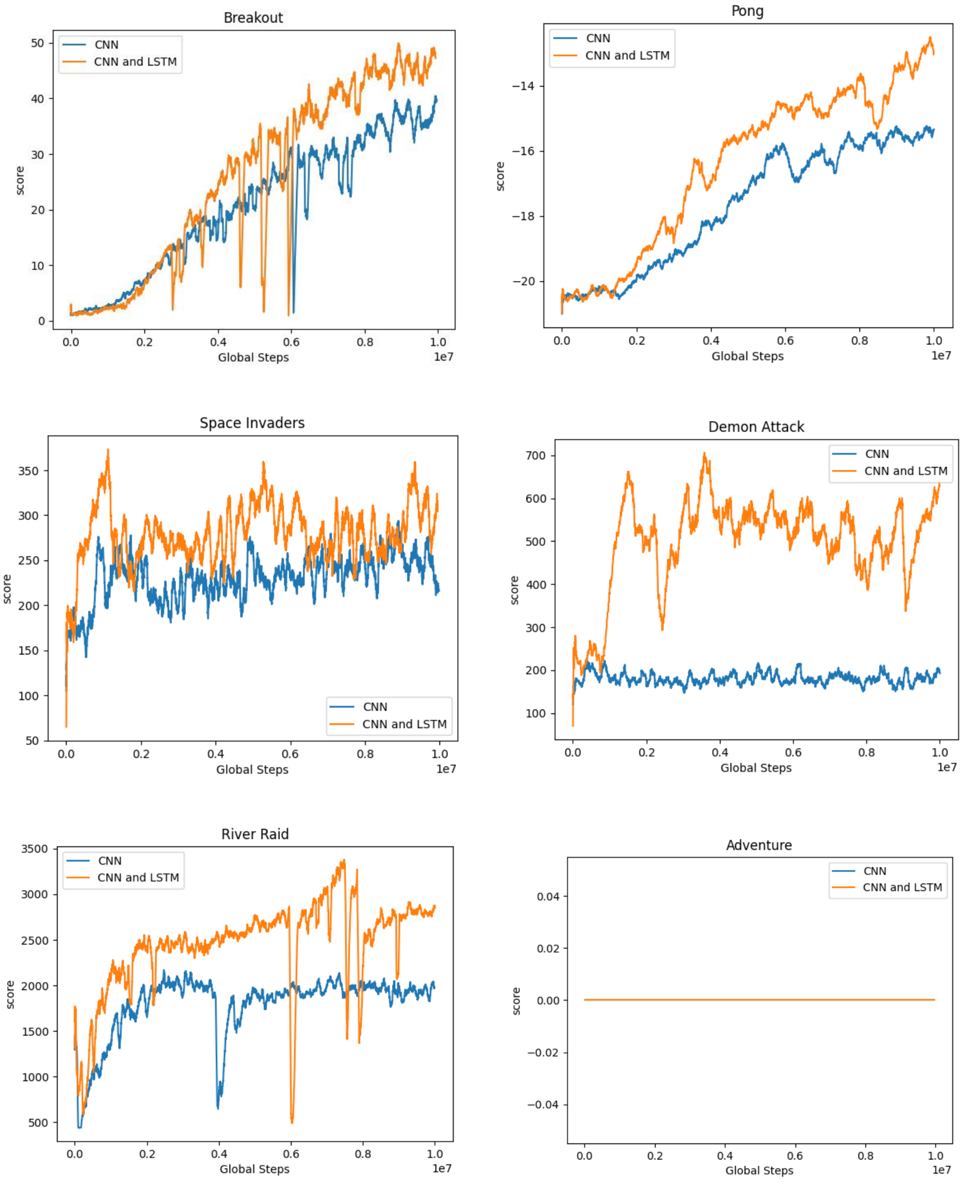


Figure A.1. Data Efficiency Comparison of Different Network Structures of A3C on Six Atari Game

The graph presents a comprehensive depiction of the data, illustrating the relationship between global steps and average scores (Rewards) on the x-axis and y-axis, respectively. Each line represents the average rewards achieved by four workers in each game. Notably, the graph highlights that CNN and LSTM models outperform CNN model, displaying superior results.

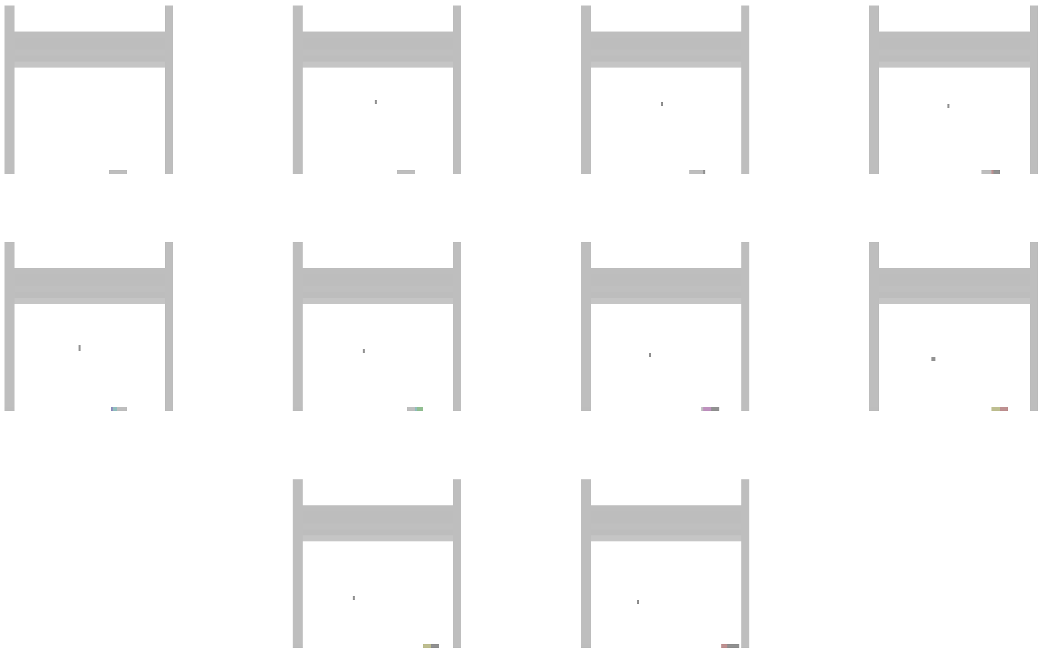


Figure A.2. Preprocessing Images from S_0 to S_9

Table A.1. The Value of Advantage, Probability of Action in Time *t*, Actual value of State, Expected Value of State, Policy Loss, Value Loss

State	R	Action	P(a,s,s')	A(a,s,s')	R(s,a)	V(s)	Entropy	P_loss	V_loss
S_0	0.0	1	0.28770602	-0.0384087614	0.586127	0.62453526	1.3756516	0.061607	0.00147523
S_1	0.0	2	0.19830747	-0.0131703071	0.592047	0.6052173	1.3771493	0.035080	0.0001734
S_2	0.0	3	0.2742222	-0.0273566702	0.598027	0.6253839	1.376347	0.049158	0.0007484
S_3	0.0	1	0.29054636	0.0208315830	0.604068	0.58323634	1.3785307	-0.011968	0.00043396
S_4	0.0	2	0.20503247	0.014649957	0.610169	0.59551966	1.3779812	-0.009434	0.0002146
S_5	0.0	0	0.24531317	-0.1455376942	0.272411	0.417949	1.3788171	0.2183001	0.021181
S_6	0.0	1	0.2502236	-0.0106026025	0.275163	0.28576553	1.3755491	0.028444	0.00011243
S_7	0.0	2	0.20908527	-0.0112971166	0.277942	0.28923947	1.3784342	0.031464	0.00012762
S_8	0.0	0	0.25524992	-0.0242292378	0.280750	0.3049791	1.3826545	0.046912	0.00058707
S_9	0.0	1	0.24490196	0.0370616692	0.283586	0.24652404	1.3782718	-0.038359	0.0013735
S_{10}	0.0	2	0.17443842	0.0666824331	-1.024494	-1.091176	1.3470733	-0.102969	0.0044649
S_{11}	0.0	1	0.18196826	0.0059162320	-1.034842	-1.0407583	1.3494589	0.0034138	0.000035
S_{12}	0.0	2	0.1927983	0.0019876067	-1.045295	-1.0472826	1.3493869	0.0102220	0.0000039
S_{13}	0.0	2	0.20588069	-0.045799126	-1.055854	-1.010054	1.3638735	0.0860224	0.0020976
S_{14}	0.0	3	0.3011563	-0.0271584070	-1.066519	-1.0393603	1.3621404	0.0462149	0.00073762
S_{15}	0.0	3	0.39521444	0.0577184122	-1.362958	-1.420676	1.3055665	-0.040526	0.0033314
S_{16}	0.0	3	0.3968141	0.0653038127	-1.376725	-1.4420286	1.3040133	-0.047319	0.0042645
S_{17}	0.0	3	0.40579394	0.0697912562	-1.390631	-1.4604224	1.3016009	-0.049929	0.00487073
S_{18}	0.0	2	0.14720367	0.0250402171	-1.404678	-1.4297181	1.3059382	-0.034916	0.00062697
S_{19}	0.0	2	0.14923386	0.0394427073	-1.418867	-1.4583093	1.3083229	-0.061946	0.00155567

Table A.2. Bootstraps from Last State

Bootstraps(V(s))	V(s)
S_5	0.616332
S_{10}	0.286450
S_{15}	-1.0772919
S_{20}	-1.43319899

$$R = \begin{cases} 0 & \text{for terminal} \\ V(s_t) & \text{for non-terminal// Bootstraps} \end{cases}$$

Action number from Policy Network:

$$a = \pi_\theta(a|s) = Neural\ Network(input : s, Weights : \theta)$$

Probability of each action from Policy Network:

$$P(\mathrm{s_{t+1}} = \mathrm{s'}|\mathrm{s_t} = \mathrm{s}, \mathrm{a_t} = \mathrm{a}) = \text{transition model for each action}$$

The value of each state in time t from the sequence of states:

$$R(s_t = s, a_t = a) = Q(s,a) = \sum_{i=0}^{k-1} \gamma^i r_{t+1} + \gamma^k V(s_{t+k}; r_{t+1})$$

For Reward (batch-5)- s_4, s_3, s_2, s_1, s_0:

$$R(s_4, a_4 = 2) = r + \gamma * V(s_5) = 0.0 + 0.99 * 0.616332 = 0.610169$$

$$R(s_3, a_3 = 1) = r + \gamma * V(s_4) = 0.0 + 0.99 * 0.610169 = 0.604067$$

$$R(s_2, a_2 = 3) = r + \gamma * V(s_3) = 0.0 + 0.99 * 0.604067 = 0.598026$$

$$R(s_1, a_1 = 2) = r + \gamma * V(s_2) = 0.0 + 0.99 * 0.598026 = 0.592046$$

$$R(s_0, a_0 = 1) = r + \gamma * V(s_1) = 0.0 + 0.99 * 0.592046 = 0.586126$$

The advantage between actual reward and expected value from value network:

$$Advantage: A = Q(s,a) - V(s)$$

For Advantage (batch-5)- s_4, s_3, s_2, s_1, s_0

$$\mathrm{A(s_0, a_0) = Q(s_0, a_0) - V(s_0)} = 0.586126 - 0.62453526 = -0.038409$$

$$\mathrm{A(s_1, a_1) = Q(s_1, a_1) - V(s_1)} = 0.592046 - 0.6052173 = -0.0131713$$

$$\mathrm{A(s_2, a_2) = Q(s_2, a_2) - V(s_2)} = 0.598026 - 0.6253839 = -0.0273579$$

$$\mathrm{A(s_3, a_3) = Q(s_3, a_3) - V(s_3)} = 0.604067 - 0.58323634 = 0.0208301$$

$$\mathrm{A(s_4, a_4) = Q(s_4, a_4) - V(s_4)} = 0.610169 - 0.59551966 = 0.0146493$$

For balancing between the exploitation and exploration of agent:

$$Entropy = E(s) = -\sum_{i=0}^{k} P(s, a_i) * \ln P(s, a_i)$$

For Entropy (batch-5)- s_4, s_3, s_2, s_1, s_0:

$$E(s_0) = -\sum_{i=0}^{4} P(s_0, a_i) * \ln P(s_0, a_i) = 1.3756516$$

$$E(s_1) = -\sum_{i=0}^{4} P(s_1, a_i) * \ln P(s_1, a_i) = 1.3771493$$

$$E(s_2) = -\sum_{i=0}^{4} P(s_2, a_i) * \ln P(s_2, a_i) = 1.376347$$

$$E(s_3) = -\sum_{i=0}^{4} P(s_3, a_i) * \ln P(s_3, a_i) = 1.3785307$$

$$E(s_4) = -\sum_{i=0}^{4} P(s_4, a_i) * \ln P(s_4, a_i) = 1.3779812$$

For Policy Loss (batch-5)- s_4, s_3, s_2, s_1, s_0:

$$Loss_{policy} = -(\ (Q(s,a) - V(s)) * \ln P(s,a) + reg * E(s))$$

$$loss(s_0) = -(-0.0384087 * ln(0.28770602) + 0.01 * 1.3756516 = 0.061606$$

$$loss(s_1) = -(-0.0131703 * ln(0.19830747) + 0.01 * 1.3771493 = 0.035080$$

$$loss(s_2) = -(-0.02735667 * ln(0.2742222) + 0.01 * 1.3763470 = 0.0491580$$

$$loss(s_3) = -(0.02083158 * ln(0.29054636) + 0.01 * 1.3785307) = -0.011962$$

$$loss(s_4) = -(0.01464996 * ln(0.20503247) + 0.01 * 1.3779812) = -0.009434$$

$$Total\_Policy\_Loss = \sum_{i=0}^{4} loss(s_i) = 0.12448$$

For Value Loss (batch-5)- s_4, s_3, s_2, s_1, s_0:

$$Loss_{value} = (Q(s,a) - V(s))^2$$

$$loss(s_0) = (A(s_0, a_0))^2 = (-0.0384087614)^2 = 0.00147523$$

$$loss(s_1) = (A(s_1, a_1))^2 = (-0.0131703071)^2 = 0.00017345$$

$$loss(s_2) = (A(s_2, a_2))^2 = (-0.0273566702)^2 = 0.00074838$$

$$loss(s_3) = (A(s_3, a_3))^2 = (0.0208315830)^2 = 0.000433955$$

$$loss(s_4) = (A(s_4, a_4))^2 = (0.014649957)^2 = 0.0002146212$$

$$Total\_Value\_Loss = \sum_{i=0}^{4} loss(s_i) = 0.0030456362$$

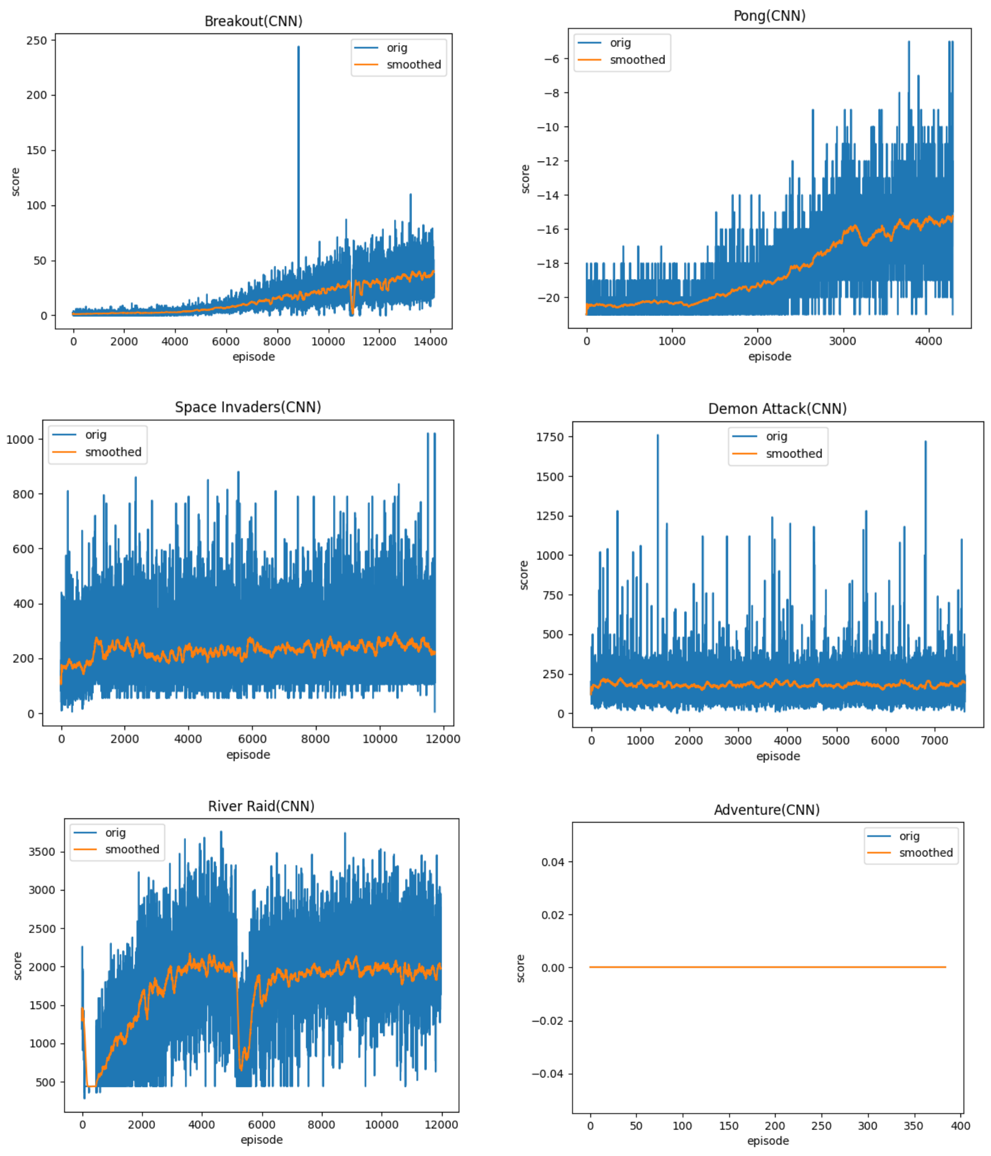


Figure A.3. Data efficiency comparison of different episode per rewards of A3C on Six Atari Game

The graph exhibits valuable insights, with the x-axis representing episodes and the y-axis displaying the average score and individual rewards. Each orange line corresponds to the average rewards attained by four workers in each game. Notably, the blue line specifically captures the rewards from each worker utilizing the CNN network structure. This distinction allows for a more detailed analysis of the performance of the CNN model across different episodes.

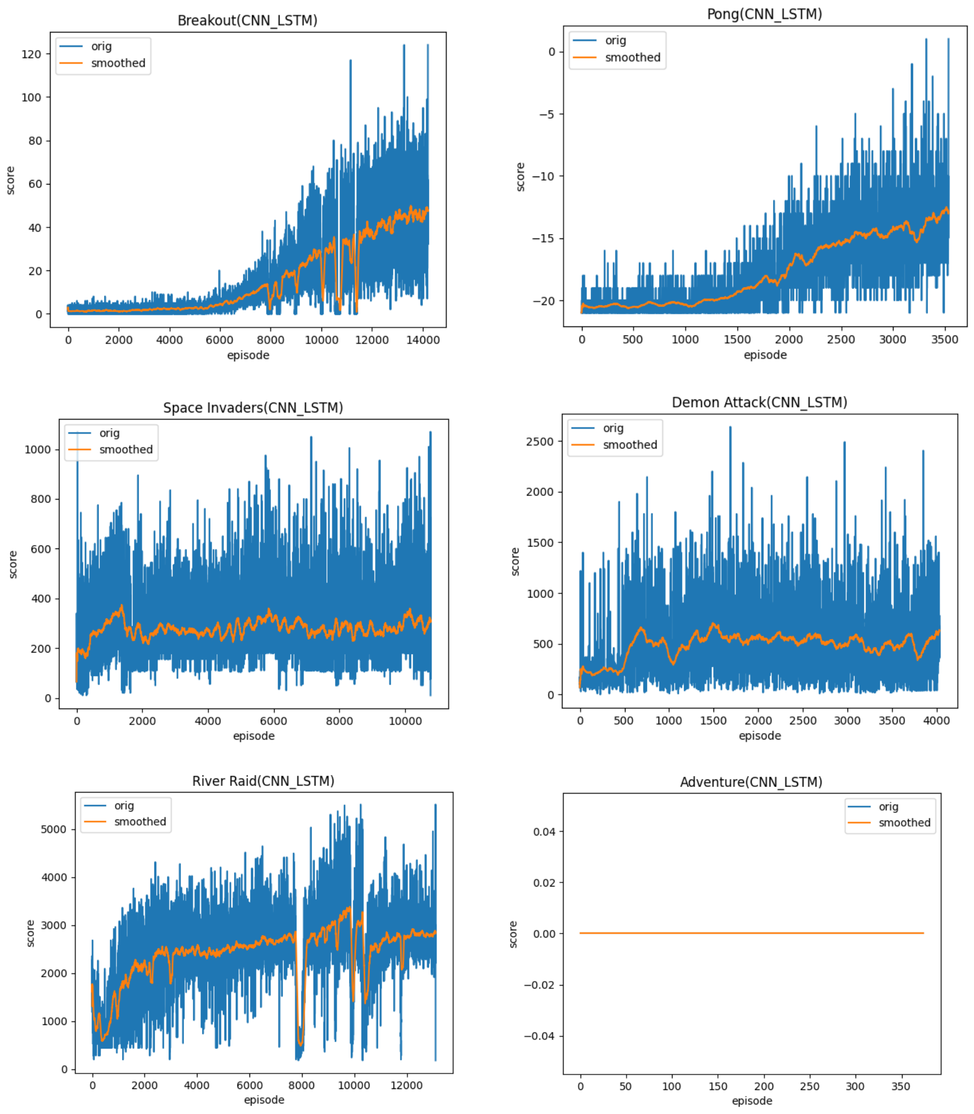


Figure A.4. Data efficiency comparison of different episode per rewards of A3C on Six Atari Game

The graph provides a comprehensive visualization of the data, where the x-axis represents episodes, and the y-axis showcases the average score and individual rewards. The orange line depicts the average rewards achieved by four workers in each game. However, the blue line specifically represents the rewards attained by each worker using CNN and LSTM network structures. This distinction allows for a more nuanced examination of the performance of the CNN and LSTM models across different episodes, enabling a deeper understanding of their individual contributions.

Algorithm Asynchronous Advantage Actor-Critic - pseudocode for each actor-learner

```
// Assume global shared parameter vectors $\theta'$ and $\theta'_v$ and global shared counter G = 0
// Assume thread-specific parameter vectors $\theta$ and $\theta_v$
Initialize thread step counter step ← 1
**repeat**
    Reset gradients:$d\theta' \leftarrow 0$ and $d\theta'_v \leftarrow 0$
    Synchronize thread-specific parameters $\theta = \theta'$ and $\theta_v = \theta'_v$
    $step_{start} = step$
    Get state s
    **repeat**
        Perform $a_t$ according to policy $\pi(a_t|s_t;\theta)$
        Receive reward $r_t$ and new state $s_{t+1}$
        $step \leftarrow step + 1$
        $G \leftarrow G + 1$
    until terminal $s_t$ or $step - step_{start} = step_{max}$

    R= { 0          for terminal
       { V($s_t$)   for non-terminal// Bootstraps

    for k ∈ $\{step - 1, \cdots, step_{start}\}$ do
        $R = r_k + \gamma R$
        Accumulate gradients $\theta$:
            $d\theta' \leftarrow d\theta' + \nabla_\theta log\, \pi(a_k|s_k;\theta)(R - V(s_k;\theta))$
        Accumulate gradients $\theta_v$:
            $d\theta'_v \leftarrow d\theta'_v + \partial(R - V(s_k;\theta))^2/\partial\theta$
    End for
    Perform asynchronous update of $\theta'$ using $d\theta'$ and of $\theta'_v$ using $d\theta'_v$
**util $G > G_{max}$**
```

Figure A.5. Algorithm Asynchronous Advantage Actor-Critic