AN EXPERIMENT OF ATTENTION-BASED GUIDED REINFORCEMENT LEARNING IN THE GAME OF SPACE INVADERS

UNDERGRADUATE THESIS

ARSYI SYARIEF AZIZ
H071191003

INFORMATION SYSTEMS UNDERGRADUATE PROGRAM

DEPARTMENT OF MATHEMATICS

FACULTY OF MATHEMATICS AND NATURAL SCIENCES

UNIVERSITAS HASANUDDIN

MAKASSAR

MARCH, 2023

AN EXPERIMENT OF ATTENTION-BASED GUIDED REINFORCEMENT LEARNING IN THE GAME OF SPACE INVADERS

UNDERGRADUATE THESIS

Submitted in partial fulfillment of the requirements for the degree of Bachelor of Computer Science from the Information Systems Undergraduate Program, Department of Mathematics, Faculty of Mathematics and Natural Sciences, Universitas Hasanuddin

ARSYI SYARIEF AZIZ
H071191003

INFORMATION SYSTEMS UNDERGRADUATE PROGRAM

DEPARTMENT OF MATHEMATICS

FACULTY OF MATHEMATICS AND NATURAL SCIENCES

UNIVERSITAS HASANUDDIN

MAKASSAR

MARCH 2023

STATEMENT OF ORIGINALITY

I hereby declare that the thesis titled:

An Experiment of Attention Based Guided Reinforcement Learning in the Game of Space Invaders

is my own work, except where due reference is made by the correct citation. I also declare that, to the best of my knowledge and belief, the work has not previously been published in any shape or form.

Jakarta, March 14, 2023

SEPULUH RIBU RUPIAH 10000 TGL. METERAI TEMPEL EE9DFAKX479425951

[illegible]

Arsyi Syarief Aziz

H071191003

AN EXPERIMENT OF ATTENTION BASED GUIDED REINFORCEMENT LEARNING IN THE GAME OF SPACE INVADERS

Approved by:

Primary Advisor	**Assistant Advisor**
[illegible]	[illegible]
Dr.Eng. Armin Lawi, S.Si., M.Eng.	**Dr. Risman Adnan Mattotorang**
NIP. 197204231995121001	**NIP. 3276062005750006**

Head of Study Program

[illegible]

Dr. Hendra, S.Si., M.Kom.
NIP. 197601022002121001

On June 6, 2023

LEGALIZATION

The thesis submitted by

Name	:	Arsyi Syarief Aziz
Student ID	:	H071191003
Study Program	:	Information Systems
Thesis Title	:	An Experiment of Attention-Based Guided Reinforcement Learning in the Game of Space Invaders

Was successfully defended in front of the examining committee and has been accepted in partial fulfillment for the degree of Bachelor Computer Science from the Information Systems Undergraduate Program, Faculty of Mathematics and Natural Sciences, Universitas Hasanuddin

EXAMINING COMMITTEE

Chair	: Dr.Eng. Armin Lawi, S.Si., M.Eng.	([illegible])
Secretary	: Dr. Risman Adnan Mattotorang	([illegible])
Member	: Dr. Hendra, S.Si., M.Kom.	([illegible])
Member	: Andi Muhammad Amil Siddik, S.Si., M.Si.	([illegible])

Approved in : Makassar

Date : June 6, 2023

KEMENTERIAN PENDIDIKAN DAN KEBUDAYAAN UNIVERSITAS HASANUDDIN DEPARTEMEN MATEMATIKA FAKULTAS MIPA

ACKNOWLEDGEMENTS

Writing a thesis can be a long and arduous journey, but it is made much easier when you have the support of amazing people who believe in you. As such, I would like to take a moment to express my heartfelt appreciation to all those who have helped me along the way.

First and foremost, I would like to acknowledge my advisors, Dr. Armin Lawi and Dr. Risman Adnan for their unwavering support, guidance, and boundless knowledge which have been critical in shaping my research. Their insightful suggestions have helped me to refine my thesis, making it a comprehensive and cohesive body of work.

I would also like to extend my deepest gratitude to my examiners, Dr. Hendra and Bapak Amil Siddik. Their invaluable feedback and constructive criticism have helped me to strengthen the structure of my thesis.

Of course, I cannot forget to thank my amazing family, especially my parents and siblings. Their unwavering love, encouragement, and support have kept me going through the tough times and provided me with the motivation to keep pushing forward. Without their support and belief in me, I may not have been able to complete my thesis on time.

Last but not least, I want to thank all my friends, colleagues, and academic peers who have provided me with encouragement throughout my journey. Their support have made the journey much more enjoyable and fulfilling.

Jakarta, March 14, 2023

[illegible]

Arsyi Syarief Aziz

THESIS PUBLICATION APPROVAL FOR ACADEMIC PURPOSES

As an academic member of Universitas Hasanuddin, I, the undersigned:

Name	: Arsyi Syarief Aziz
Student ID	: H071191003
Study Program	: Information Systems
Department	: Mathematics
Faculty	: Faculty of Mathematics and Natural Sciences

approve, in the interest of scientific development, to grant Universitas Hasanuddin the non-exclusive royalty-free right to my work titled:

An Experiment of Attention-Based Guided Reinforcement Learning in the Game of Space Invaders.

As per the approval granted, I acknowledge and agree that the university has the right to save, transfer media/format, manage in a database, maintain, and publish my thesis as long as it includes my name as the author/creator and as the copyright owner.

This approval is made in good faith and is to be used as appropriate.
Approved in Jakarta on the 14th day of March, 2023.

The Declarer,

[illegible]

(Arsyi Syarief Aziz)

ABSTRACT

This research aims to imitate three competencies of human-level general intelligence: perception, learning, and attention. The objective of the study was to investigate the effects of visual attention mechanisms on guided-reinforcement learning models. The methodology for the study involved performing two experiments in the Space Invaders game. The first experiment implemented two guided reinforcement learning algorithms and assessed them based on their acquired scores to obtain the most suitable model for the environment. The second experiment then integrated the obtained best model with three types of visual-attention modules and evaluated the modules based on their acquired scores and a visual analysis of their saliency maps. The results demonstrated that the visual attention mechanisms improved the guided-reinforcement learning models' ability to focus on objects in the environment. However, further research is needed to establish performance improvements over non-attention models.

Keywords: *Reinforcement Learning, Imitation Learning, Visual Attention, Generative Adversarial Network, Deep Learning.*

ABSTRAK

Penelitian ini bertujuan untuk meniru tiga kompetensi kecerdasan umum manusia: persepsi (*perception*), pembelajaran (*learning*), dan perhatian (*attention*). Objektif penelitian ini adalah untuk menyelidiki efek mekanisme *visual attention* pada model *guided reinforcement learning*. Metodologi penelitian ini melibatkan dua eksperimen pada permainan Space Invaders. Eksperimen pertama mengimplementasikan dua algoritme *guided reinforcement learning* dan mengevaluasinya berdasarkan skor untuk memperoleh model yang paling sesuai untuk lingkungan tersebut. Setelah itu, eksperimen kedua mengintegrasikan model terbaik yang diperoleh dengan tiga jenis modul *visual attention* dan mengevaluasi modul-modul tersebut berdasarkan skor yang diperoleh dan analisis visual dari peta saliensinya. Hasil penelitian menunjukkan bahwa mekanisme *visual attention* meningkatkan kemampuan model *guided reinforcement learning* untuk fokus pada objek-objek di lingkungan. Namun, penelitian lebih lanjut diperlukan untuk menunjukkan peningkatan kinerja dibandingkan dengan model-model tanpa *visual-attention*.

Kata kunci: *Reinforcement Learning, Imitation Learning, Visual Attention, Generative Adversarial Network, Deep Learning*

Judul	:	An Experiment of Attention-Based Guided Reinforcement Learning in the Game of Space Invaders
Nama	:	Arsyi Syarief Aziz
NIM	:	H071191003
Program studi	:	Sistem Informasi

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I
INTRODUCTION

1.1 Background of Study

Recent advancements in artificial intelligence have demonstrated its capability of achieving ”superhuman” performance in numerous complicated tasks. For example, artificial intelligence has demonstrated the ability to defeat grandmasters in tournaments such as chess, shogi, and Go (Silver et al., 2016; 2017). Furthermore, in a more recent development, it has also demonstrated the ability to generate hyper-realistic images from text data (Ramesh et al., 2022), which in the past was only possible through an artist’s rendition.

Currently, one particular long-term goal of much artificial intelligence research is to develop a unified framework known as artificial general intelligence (AGI). AGI contributes to the ability of machines to comprehend, learn, and act intelligently in all tasks that humans can do. One way to achieve this goal of developing AGI is to imitate the competencies of human-level general intelligence, including, but not limited to, perception, learning, and attention (Goertzel, 2014). Moreover, in learning, machines are required to have the ability to imitate other’s behavior and to reinforce previous positive and negative experiences.

Two machine learning paradigms support the competencies stated above: reinforcement learning and imitation learning. In reinforcement learning, a machine learns how to solve a problem through self-exploration. Meanwhile, in imitation learning, a machine learns by imitating others’ behavior.

Previous studies have combined reinforcement learning and imitation learning (Hester et al., 2018; Kang et al., 2018) to create the guided-reinforcement learning approach. In guided-reinforcement learning, the machine not only learns the actions of a demonstrator, but it also can explore on its own. The studies showed improved performance over reinforcement learning and imitation learning alone.

However, studies on guided reinforcement learning have only shown its capability to perform perception and learning, and they are yet to incorporate it with attention. Therefore, this research experiments to integrate visual-attention mechanisms with guided-reinforcement learning models to fill the research gap.

1.2 Research Questions

Based on the background mentioned above, the researcher asks two important questions:

1. How can visual-attention mechanisms be used to improve guided reinforcement learning models?
2. What effect do visual-attention mechanisms have on guided reinforcement learning models?

1.3 Research Objectives

By exploring the research questions, the objectives of this research are:

1. To investigate the effects of visual-attention mechanisms on guided reinforcement learning models.
2. To combine visual-attention mechanisms with guided reinforcement learning models.

1.4 Research Significance

This research explores the feasibility of incorporating visual-attention mechanisms into guided reinforcement learning models to simultaneously imitate three competencies of human-level general intelligence: perception, learning, and attention.

1.5 Overview of Structure

This thesis is structured as follows: introduction/background of study; review of literature; research methodology; results and discussions; and conclusion and recommendations.

CHAPTER II
LITERATURE REVIEW

This literature review covers six topics related to this research. The chapter will begin with an introduction to reinforcement learning, including its foundational concepts and several of the most notable model-free reinforcement learning algorithms. After that, the chapter will provide an overview of deep reinforcement learning by explaining its building blocks. Next, the chapter will discuss what generative adversarial networks are and a few of their stabilization techniques. Then, the chapter will explain guided reinforcement learning, along with its algorithms. After that, the chapter will explain the visual attention mechanisms employed in this research. To conclude, the chapter will review previous implementations of visual-attention in the deep learning literature.

2.1 Reinforcement Learning Foundations

Reinforcement learning is an approach to machine learning that involves the interaction between a learner, known as the agent, and its surroundings, the environment. The goal of the agent in this approach is to learn how to behave in the environment which it achieves though a process of trial and error, receiving rewards and punishments for certain actions. Through reinforcement learning, the agent learns to take the most beneficial actions in order to maximize its rewards.

This section will explain how reinforcement learning occurs through six subtopics. Firstly, it will describe the Markov decision process and explain how it relates to reinforcement learning. Then it will formulate the definition of rewards and returns. After that, it will describe what a policy means in reinforcement learning. Next, it will explain what value functions are and the Bellman equations that relate to it. Finally, it will describe several model-free reinforcement learning algorithms.

2.1.1 Markov Decision Process

The Markov decision process (MDP) is a mathematical framework used to describe reinforcement learning problems. This framework models the environment as a stochastic process, where the agent can influence the environment's behavior through a selected action (Boutilier et al., 1999, p. 3; Puterman, 2005, p. 17), which determines the environment's next states (Boutilier et al., 1999, p. 3). In addition, the environment also provides a reward signal to the agent, that the agent learns to maximize through its interactions with the environment (Sutton & Barto, 2018, p. 48).

More specifically, the MDP can be described as a 4-tuple $\langle \mathcal{S}, \mathcal{A}, P, R \rangle$, which consists of a set of states, $\mathcal{S}$, a set of actions, $\mathcal{A}$, a transition probability function, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, and a reward function, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ (Kang et al., 2018; Schulman et al., 2015; Puterman, 2005, p. 20). The interaction of the agent and its environment occurs at a sequence of discrete time-steps ($t = 0, 1, 2, \ldots$). For each time-step, t, the agent observes a state, $s_t \in \mathcal{S}$, of the environment and performs an action, $a_t \in \mathcal{A}$. At the proceeding time step, $t+1$, responding to the performed action, the environment gives a reward signal, $r_{t+1} = R(a_t, s_t)$, to the agent, and updates its state to $s_{t+1} \in \mathcal{S}$ based on the transition probability distribution, $P(\cdot|s_t, a_t)$. This processes repeats until the final timestep, T, to which a sequence of interactions is obtained known as a trajectory, $\tau = (s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, \ldots)$.

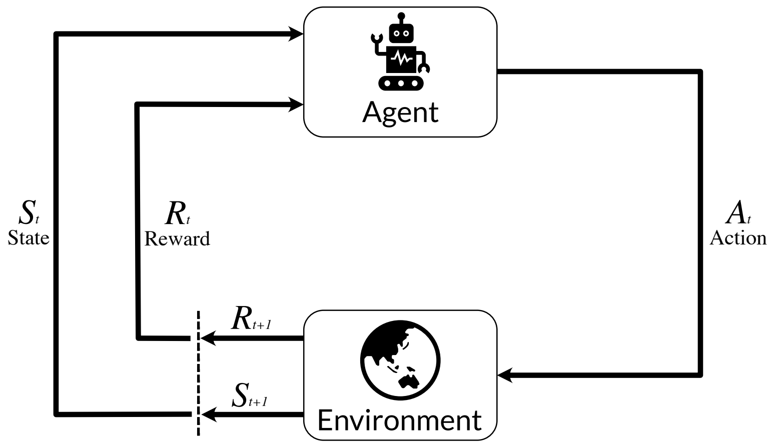


Figure 2.1: *A visual representation of the agent-environment interaction in the Markov decision process (adapted from Sutton and Barto, 2018).*

2.1.2 Rewards

Rewards are the values that the agent receives as a result of its interactions with the environment. These values are used by the environment to communicate to the agent what it wants to be achieved (Sutton & Barto, 2018, p. 53). A positive reward value indicates that the goals of the problem are being achieved, while a negative value indicates the goals are not being achieved. By receiving rewards, the agent learns which actions are beneficial and which are not in reaching its goal.

2.1.3 Returns

Returns refer to the total amount of rewards that an agent accumulates throughout its interactions with the environment. In episodic tasks that have a finite sequence of time steps T, returns can simply be calculated as the summation of all rewards received by the agent (Sutton & Barto, 2018):

$$G_t = r_{t+1} + r_{t+2} + \cdots + r_T, \tag{2.1}$$

where r_t represents the reward obtained at time step t.

For non-episodic tasks, with an infinite sequence of time steps, i.e. when $T = \infty$, a discount factor $\gamma \in [0, 1)$ is introduced to induce convergence to the return calculation. The returns for non-episodic tasks is formulated as:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{k+t+1}. \tag{2.2}$$

Equivalently, equation 2.2 can also be applied to the episodic task formulation by replacing the summation to a finite number of terms. This type of formulation is known as discounted returns.

The addition of a discount factor in the discounted returns allows agents to view the environment in a near-sighted or far-sighted view of rewards (Sutton & Barto, 2018, p. 55). That is, when γ is close to 0, the agent will have a near-sighted view on the rewards and prefer rewards acquired close to the present. Meanwhile, when γ is close to 1, the agent will have a far-sighted view of rewards and will consider rewards acquired in the future.

2.1.4 Policy

An agent's behavior is determined by a set of mathematical rules, known as the policy. These mathematical rules map an agent's observed state to a corresponding action. Achiam (2018) explains that there are two ways to formulate the rules: deterministically and stochastically. In the deterministic setting, the rules are referred to as deterministic policies which are functions, $\mu(s)$, that map a state, s, to a deterministic action, a. On the other hand, in the stochastic setting, the rules are known as stochastic policies which are functions that map an observed state, s, to a probability distribution, $\pi(\cdot|s)$, of all possible actions under the given state. In a stochastic policy, the selected action is obtained by sampling an action from the induced probability distribution.

2.1.5 Value Functions

The value functions are used to describe how well an agent is expected to perform in the environment. Sutton and Barto (2018) explains that there are two types of value functions: the state-value function and the action-value function. The state-value function, $V^\pi(s)$, defines the amount of rewards that an agent is expected to obtain when starting from state, s, and thereafter always acting under the policy, π. It is formulated as:

$$V^\pi(s) = \mathbb{E}_\pi\left[G_t | s_0 = s\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{k+t+1} \middle| s_t = s\right], \tag{2.3}$$

where t is the current timestep and k are the future timesteps. Likewise, the action-value function, $Q^\pi(s,a)$, defines the expected rewards of performing an action, a, at a state, s, and then following the policy, π:

$$Q^\pi(s,a) = \mathbb{E}_\pi\left[G_t | s_0 = s, a_0 = a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k r_{k+t+1} \middle| s_t = s, a_t = a\right]. \tag{2.4}$$

Furthermore, the state-value and action-value functions can be combined to define another value function known as the advantage, $A^\pi(s,a)$ (François-Lavet et al., 2018, p. 238). The advantage describes the addition (or reduction) of rewards that an agent is expected to acquire when taking action, a, at a state, s, instead of following the action induced by the policy π:

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s). \tag{2.5}$$

2.1.6 Bellman Equations

The Bellman equations are value functions written in recursive form. They are an essential formulation for some reinforcement learning algorithms because they enable the use of dynamic programming. For the state-value function, its Bellman equation is defined as:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi\left[r_{t+1} + \sum_{k=1}^{\infty} \gamma^k r_{k+t+1} \middle| s_t = s\right] \\ &= \mathbb{E}_{a\sim\pi, s'\sim P(\cdot|s,a)}\left[r_{t+1} + \gamma V^\pi(s') | s_t = s\right]. \end{aligned} \tag{2.6}$$

Meanwhile, for the action-value function, its Bellman equation is defined as (Achiam, 2018):

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbb{E}_{\pi}\left[r_{t+1} + \sum_{k=1}^{\infty} \gamma^k r_{k+t+1} \middle| s_t = s \right] \\ &= \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[r_{t+1} + \mathbb{E}_{a\sim\pi} \left[Q^{\pi}(s',a) \right] | s_t = s, a_t = a \right]. \end{aligned} \tag{2.7}$$

In these equations, s' is a future state sampled from the transition probability function, $P(\cdot|s,a)$.

2.1.7 Model-Free Reinforcement Learning

There are two methods that can be used to solve reinforcement learning problems: model-based and model-free reinforcement learning. In model-based reinforcement learning, a known (Anthony et al., 2017; Silver et al., 2017) or learned model (Ha & Schmidhuber, 2018; Nagabandi et al., 2018; Sutton, 1991) of the environment is used to plan a sequence of actions. Meanwhile, in model-free reinforcement learning, the agent learns purely from trial and error.

This section will explain about the model-free methods which are used throughout this research. More specifically, this section reviews two classes of model-free reinforcement learning: temporal difference and policy gradient.

2.1.7.1 Temporal Difference Learning

Temporal difference (TD) is a method of model-free reinforcement learning that applies dynamic programming and the Monte Carlo methods to solve reinforcement learning problems. In this formulation, TD exploits the recursive property of the Bellman equations to approximate the value functions. These estimates are updated using a method known as bootstrapping, which involves using an estimate of the value function to update the value function itself (Sutton & Barto, 2018).

2.1.7.2 One-step temporal difference learning

The simplest form of the TD method is known as one-step temporal difference learning, TD(0), and it approximates the state-value function. In this method, the learned state value function, $V(s)$, is defined by a look-up table where each entry in the table describes a mapping from a state to a state-value estimate.

The look-up table in TD(0) is updated iteratively based on the agent's interaction with its environment. For each state that an agent encounters in the interaction, TD(0) observes the acquired reward, r_{t+1}, and then updates the state's value in the table through the update rule:

$$V(s_t) \leftarrow V(s_t) + \alpha \left[r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \right], \tag{2.8}$$

where α is the step size that controls how fast the algorithm learns, and γ is the discount factor used to weigh the importance of future rewards.

2.1.7.3 Q-Learning

Q-learning (C. J. C. H. Watkins, 1989) is a variant of the TD method which approximates the optimal action-value function, $Q^*(s, a)$. Like TD(0), Q-learning implements a look-up table to define the learned action-value function, $Q(s, a)$. The method learns $Q(s, a)$ by selecting the maximizing action for the next state, s_{t+1}, in the Bellman equation (2.7), and then updating the corresponding entry in the table:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]. \tag{2.9}$$

An interesting property of Q-learning is that if $Q^*(s,a)$ is known, then the optimum policy, $\mu^*(s)$, can be found simply by selecting the maximizing action for each state:

$$\mu^*(s) = \max_a Q^*(s, a). \tag{2.10}$$

This property, however, cannot be directly exploited since the look-up table in Q-learning provides only an approximation of the optimum action-value function, $Q^*(s, a)$. As a result, the estimated values might lead to exploration issues (Sutton & Barto, 2018, p. 29). To address this issue, a method known as ϵ-greedy is utilized, which allows for exploration:

$$\mu(s) = \begin{cases} \max_a Q(s, a) & \text{with probability } 1 - \epsilon \\ \text{randomly select } a \in \mathcal{A} & \text{with probability } \epsilon \end{cases} \tag{2.11}$$

In Q-learning with ϵ-greedy, the agent selects the maximizing action with a probability of $1 - \epsilon$, and selects a random action with a probability of ϵ.

2.1.7.4 Deep Q-Learning

Deep Q-Learning (Mnih et al., 2013), DQN, is a variant of Q-learning which approximates the mappings of state-action pairs to action values through a deep neural network (see Section 2.2). This approximation helps reduce the memory requirements of Q-learning. In addition to using a neural network, DQN implements a replay buffer, which stores previous interactions used to update the policy in an off-policy way (see Mnih et al., 2013).

2.1.7.5 Policy Gradient

As described in the previous subsection, temporal difference methods learn a policy through value function approximations. These methods work well and have even been proven by C. J. Watkins and Dayan (1992). However, Sutton et al. (1999) explains that the temporal difference methods have some fallbacks: (1) the policy in temporal difference learning is usually deterministic, which does not reflect the usually stochastic optimum policy, and (2) small changes in the approximate can cause the agent to select sub-optimal actions.

An alternate approach to reinforcement learning is to directly learn an approximation of the optimum policy through a policy gradient method. In the policy gradient methods, the goal is to optimize a parameterized policy π_θ by maximizing a cost function, $J(\theta)$, defined as the state-value function evaluated at the starting state s_0. The optimization of the policy is performed using the gradients of the cost function, formulated as (see Sutton and Barto, 2018, p. 325 for derivation):

$$\nabla_\theta J(\theta) \propto \mathbb{E}_{\pi_\theta}\left[Q^{\pi_\theta}(s,a)\nabla_\theta \ln \pi_\theta(a|s)\right], \tag{2.12}$$

in methods which will be elaborated in the following subsections.

2.1.7.6 REINFORCE

REINFORCE (Williams, 1992) is a policy gradient method that directly optimizes the policy, π_θ, based on the policy gradient equation (2.12). To perform this optimization, REINFORCE utilizes the Monte Carlo methods. Let $s_t \in \mathcal{S}$ be the state encountered at time-step t and $a_t \sim \pi_\theta$ be the action sampled under the policy π_θ at time-step t. The policy gradient equation then becomes (Sutton & Barto, 2018, p. 327):

$$\begin{aligned}\nabla_\theta J(\theta) &\propto \mathbb{E}_{\pi_\theta}\left[Q^{\pi_\theta}(s,a)\nabla_\theta \ln \pi_\theta(a|s)\right]\\ &= \mathbb{E}_{\pi_\theta}\left[Q^{\pi_\theta}(s_t,a_t)\nabla_\theta \ln \pi_\theta(a_t|s_t)\right]\\ &= \mathbb{E}_{\pi_\theta}\left[G_t\nabla_\theta \ln \pi_\theta(a_t|s_t)\right] \qquad \left(\text{since } \mathbb{E}_{\pi_\theta}\left[G_t|s_t,a_t\right] = Q^{\pi_\theta}(s_t,a_t)\right).\end{aligned} \tag{2.13}$$

Using the equation above, REINFORCE works to sample a trajectory under the policy, π_θ, and for each state-action pair in the trajectory, at timestep t, it calculates

Equation 2.13. The obtained value is then used to update the policy parameters, θ, through gradient ascent:

$$\theta_{t+1} \leftarrow \theta_t + \alpha\gamma^t G_t \nabla_\theta \ln \pi_\theta(a_t|s_t), \tag{2.14}$$

where α is the learning rate and γ is the discount factor.

In the gradient ascent step above, it can be seen that the policy updates linearly with the discounted returns. That is, if the discounted returns are positive, the policy parameters will be updated in the magnitude of $\gamma^t G_t$ and make the probability of choosing the action a at state s higher for future visitations. Conversely, if the discounted returns are negative, the policy update will be updated in the opposite direction with the magnitude of $\gamma^t G_t$ and make the probability of choosing action a at state s lower for future visitations.

2.1.7.7 REINFORCE with baseline

The use of Monte Carlo methods to approximate Equation 2.13 in REINFORCE makes it prone to high variance which can make learning slow (Sutton & Barto, 2018, p. 329). One way to reduce variance is to introduce a baseline function, $b(s)$, into the policy gradient formulation (2.12) to define an extension to the policy gradient equation:

$$\nabla_\theta J(\theta) \propto \mathbb{E}_{\pi_\theta}\left[(G_t - b(s_t))\nabla_\theta \ln \pi_\theta(a_t|s_t)\right]. \tag{2.15}$$

The subtraction of $b(s)$ from G_t in the equation above is permitted since the baseline term does not change the value of the gradient. More precisely, it is because the value of the baseline quantity is zero:

$$\begin{aligned} \mathbb{E}_{a\sim\pi_\theta}[b(s)\nabla_\theta \ln \pi_\theta(a|s)] &= b(s)\sum_a \pi_\theta(a|s)\frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} \\ &= b(s)\nabla_\theta \sum_a \pi_\theta(a|s) \\ &= b(s)\nabla_\theta 1 = 0. \end{aligned} \tag{2.16}$$

The choice of baseline to reduce variance can be an arbitrary function so long as the function itself is not parameterized by a. Examples of baselines include a constant or a learned estimate of the state-value function $V(s)$. The latter is commonly used in reinforcement learning since its usage formulates the advantage, $A_t = G_t - V(s)$, which can be approximated by a parametric function $V_w(s)$ to calculate the advantage estimate, $\hat{A}_t = G_t - V_w(s)$. By substituting $b(s_t)$ with

$V_w(s_t)$ in Equation 2.15, it then becomes:

$$\begin{aligned}\nabla_\theta J(\theta) &\propto \mathbb{E}_{\pi_\theta}\left[(G_t - V_w(s))\nabla_\theta \ln \pi_\theta(a_t|s_t)\right] \\ &= \mathbb{E}_{\pi_\theta}\left[\hat{A}_t \nabla_\theta \ln \pi_\theta(a_t|s_t)\right].\end{aligned} \tag{2.17}$$

From the formulation above, it is straightforward to show that the steps of this method are to first sample a trajectory under π_θ. Then calculate the advantage estimate $\hat{A}_t = G_t - V_w(s)$. Finally, update the parameters w and θ of the value function estimate V_w and policy π_θ.

2.1.7.8 Proximal Policy Optimization

Proximal Policy Optimization (Schulman et al., 2017), also known as PPO, is an extension to REINFORCE with baselines that allows multiple optimizations to be performed on a single data sample. The algorithm achieves this by optimizing a surrogate objective function to equation 2.17, based on importance sampling:

$$J(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}_t\right],. \tag{2.18}$$

Here, π_θ refers to the current policy, while $\pi_{\theta_{\text{old}}}$ refers to the policy before the update.

An issue that occurs with directly applying multiple optimizations on the surrogate objective function, however, is that it can make the policy update too large. To prevent this from occurring, Schulman et al. (2017) introduces a constraint to equation 2.18, which constrains the allowed divergence from a previous policy to an updated policy. Let $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ denote the probability ratio of the current policy, π_θ, and the policy before the update, $\pi_{\theta_{\text{old}}}$. The algorithm constrains the surrogate objective function by performing a clipping operation defined as:

$$L_t^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right], \tag{2.19}$$

where ϵ is a hyper-parameter that controls the allowed divergence of a policy update from the previous policy.

In addition to the policy gradient loss above, PPO also introduces an entropy regularization term, $S[\pi_\theta](s_t)$, to increase exploration of the policy, as well as a squared error loss, $L^{VF}(\theta) = (V_\theta(s_t) - V_t^{\text{targ}})^2$, to train the learned value function. Together, these terms are combined to define the loss function of PPO:

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)], \tag{2.20}$$

where c_1 and c_2 are hyperparameters that control the weights of the value function and the entropy regularization, respectively. Algorithm 1 describes how PPO is trained.

Algorithm 1 Proximal Policy Optimization (Schulman et al., 2017)

```
1: for iteration = $1, 2, \dots$ do
2:     for actor = $1, 2, \dots, N$ do
3:         Run policy $\pi_{\theta_{\text{old}}}$ in the environment for $T$ timesteps.
4:         Compute advantage estimates $\hat{A}_1, \dots, \hat{A}_T$.
5:     end for
6:     Optimize surrogate $L$ w.r.t. $\theta$, with $K$ epochs and minibatch size $M \leq NT$.
7:     $\theta_{\text{old}} \leftarrow \theta$.
8: end for
```

2.2 Building Blocks of Deep Reinforcement Learning

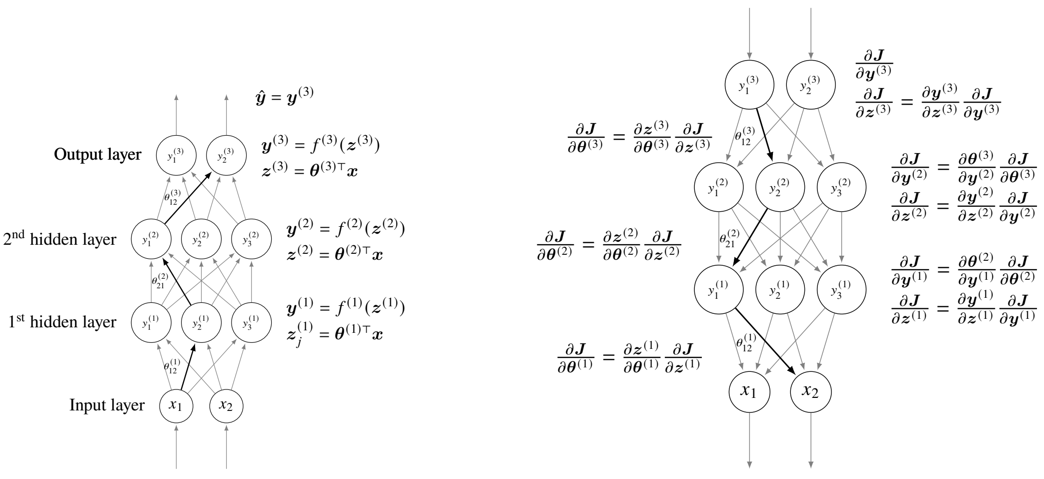
This section discusses how the policies and value functions of the agents are constructed in the perspective of deep reinforcement learning. In this discussion, two key concepts will be covered: the artificial neural network and the convolution neural network.

2.2.1 Artificial Neural Network

The artificial neural network, often abbreviated as the neural network or simply network, is the most fundamental concept in deep reinforcement learning and it can be understood as the building blocks of the agent's brain. Like a real brain, the artificial neural network consists of a network of neurons that work together to make inferences. This network is arranged in several layers. The first layer of a network is called the input layer, which serves as the entry point of an environment signal before it is processed. The middle layers are called the hidden layers, which process the signal through the chaining of non-linear mathematical calculations. The last layer is the output layer, which outputs the resulting inference.

Each neuron between two consecutive layers of a network is connected by a connection, which propagates a signal based on a weight θ. When a neuron receives a signal from a previous neuron, the neuron activates the signal using a non-linear activation function and outputs another signal. This signal is then propagated to the connected neurons at the subsequent layers until it reaches the output layer. The resulting signal at the output layer is what determines the inferences made by the network, which can include an action selection for a policy representation, or a value function estimate.

To update the network parameters, the neural network performs backpropagation based on the cost function, J, induced by the network inference. In this process, the network calculates the gradients of the cost function and propagates its value backward through the network to obtain gradients of the cost function with respect to the network parameters (see Figure 2.2b). Using these gradients, the network then performs a parameter update, either through gradient ascent, $\theta = \theta + \alpha\frac{\partial}{\partial\theta}J(\theta)$, or gradient descent, $\theta = \theta - \alpha\frac{\partial}{\partial\theta}J(\theta)$, with α as the learning rate of the network.



Output layer
2nd hidden layer
1st hidden layer
Input layer
$\hat{\boldsymbol{y}} = \boldsymbol{y}^{(3)}$
$\boldsymbol{y}^{(3)} = f^{(3)}(\boldsymbol{z}^{(3)})$
$\boldsymbol{z}^{(3)} = \boldsymbol{\theta}^{(3)\top}\boldsymbol{x}$
$\boldsymbol{y}^{(2)} = f^{(2)}(\boldsymbol{z}^{(2)})$
$\boldsymbol{z}^{(2)} = \boldsymbol{\theta}^{(2)\top}\boldsymbol{x}$
$\boldsymbol{y}^{(1)} = f^{(1)}(\boldsymbol{z}^{(1)})$
$z_j^{(1)} = \boldsymbol{\theta}^{(1)\top}\boldsymbol{x}$
$\frac{\partial J}{\partial \boldsymbol{y}^{(3)}}$
$\frac{\partial J}{\partial \boldsymbol{z}^{(3)}} = \frac{\partial \boldsymbol{y}^{(3)}}{\partial \boldsymbol{z}^{(3)}}\frac{\partial J}{\partial \boldsymbol{y}^{(3)}}$
$\frac{\partial J}{\partial \boldsymbol{\theta}^{(3)}} = \frac{\partial \boldsymbol{z}^{(3)}}{\partial \boldsymbol{\theta}^{(3)}}\frac{\partial J}{\partial \boldsymbol{z}^{(3)}}$
$\frac{\partial J}{\partial \boldsymbol{y}^{(2)}} = \frac{\partial \boldsymbol{\theta}^{(3)}}{\partial \boldsymbol{y}^{(2)}}\frac{\partial J}{\partial \boldsymbol{\theta}^{(3)}}$
$\frac{\partial J}{\partial \boldsymbol{z}^{(2)}} = \frac{\partial \boldsymbol{y}^{(2)}}{\partial \boldsymbol{z}^{(2)}}\frac{\partial J}{\partial \boldsymbol{y}^{(2)}}$
$\frac{\partial J}{\partial \boldsymbol{\theta}^{(2)}} = \frac{\partial \boldsymbol{z}^{(2)}}{\partial \boldsymbol{\theta}^{(2)}}\frac{\partial J}{\partial \boldsymbol{z}^{(2)}}$
$\frac{\partial J}{\partial \boldsymbol{y}^{(1)}} = \frac{\partial \boldsymbol{\theta}^{(2)}}{\partial \boldsymbol{y}^{(1)}}\frac{\partial J}{\partial \boldsymbol{\theta}^{(2)}}$
$\frac{\partial J}{\partial \boldsymbol{z}^{(1)}} = \frac{\partial \boldsymbol{y}^{(1)}}{\partial \boldsymbol{z}^{(1)}}\frac{\partial J}{\partial \boldsymbol{y}^{(1)}}$
$\frac{\partial J}{\partial \boldsymbol{\theta}^{(1)}} = \frac{\partial \boldsymbol{z}^{(1)}}{\partial \boldsymbol{\theta}^{(1)}}\frac{\partial J}{\partial \boldsymbol{z}^{(1)}}$

(a) ***(b)***

Figure 2.2: *A computational graph that visualizes a feed-forward neural network. For visual simplicity, the bias term is removed. (a) shows the process of forward propagation which is used to produce an inference $\hat{y}$ from an input vector x. In the figure, it can be seen that the values of the neurons at each layer (l) are obtained by multiplying the activation values of the connected neurons at the previous layer with the corresponding connection weights $\theta^{(l)}$. These values are activated through an activation function $f^{(l)}(z)$ to obtain the activation values. (b) shows the process of calculating the gradients of the network through backpropagation. This step is performed after a forward propagation step. In backpropagation, the value of a cost function, J, is calculated along with its gradient and is then backwards propagated to the previous layers of the network, through the chain rule, to obtain the gradients of the objective function w.r.t. the network parameters.*

2.2.2 Convolutional Neural Network

The convolutional neural network (CNN) is a special type of neural network that performs image processing tasks based on a special operation known as the convolution. The convolution is used to extract features from an image, such as its edges and curves. Moreover, these feature extractions can be chained together through several convolution operations to obtain higher-level features of an image, such as the features of objects or even the object itself.

In the following subsections, the three general components of the CNN will be discussed: The convolutional layer, the pooling layer, and the fully connected layer.

2.2.2.1 Convolutional Layer

The convolutional layer is a layer in the CNN that performs the convolution operation on an input image. In performing this operation, the convolution uses a set of learnable parameters known as the kernel. The kernel is defined as a matrix of a smaller spatial dimensionality than the input image, which can span several channels. Furthermore, each of these channels represent a unique matrix of learnable parameters.

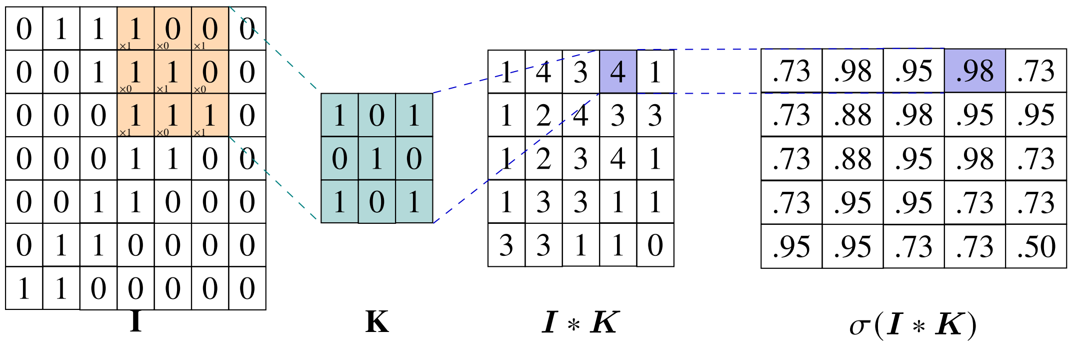


Figure 2.3: *A visualization of the convolution operation performed on a single channel image of size 7×7 with a 3×3 kernel. In this operation, the kernel slides across the input image and performs a dot product between I and K to obtain the resulting feature map. After obtaining the feature map, its values are then fed into an activation function. In this example, the activation function is the sigmoid function.*

During a convolution operation, the convolutional layer slides its kernel across an image, and for each position, it calculates the dot product between the input image and the kernel. The resulting values of these dot products make up an output image known as the feature map. In addition, the convolutional layer applies an activation function to the feature maps to provide non-linearity to the network.

2.2.2.2 Pooling Layer

After extracting the features of an input image, a pooling layer can then be implemented to downsample the feature maps. I. Goodfellow et al. (2016) explains that this downsampling operation provides two benefits to the network: (1) it reduces