U. HAIDER

ADVERTISEMENT CLICK PREDICTION USING REINFORCEMENT LEARNING

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
ATILIM UNIVERSITY

UMAIR HAIDER

A MASTER OF SCIENCE THESIS
IN
THE DEPARTMENT OF SOFTWARE ENGINEERING

JULY 2023

ATILIM UNIVERSITY 2023

ADVERTISEMENT CLICK PREDICTION USING REINFORCEMENT LEARNING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
ATILIM UNIVERSITY

BY

UMAIR HAIDER

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTERS OF SCIENCE
IN
SOFTWARE ENGINEERING

JULY 2023

Approval of the Graduate School of Natural and Applied Sciences, Atilim University.

_____

Prof. Dr. Ender KESKİNKILIÇ

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of **Masters of Science in Software Engineering Department, Atilim University**.

_____

Prof. Dr. Ali YAZICI

Head of Department

This is to certify that we have read the thesis ADVERTISEMENT CLICK PREDICTION USING REINFORCEMENT LEARNING submitted by UMAIR HAIDER and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Masters of Science.

_____

Assoc. Prof. Dr Beytullah YILDIZ

Supervisor

Examining Committee Members:

Assoc. Prof. Dr. Gürhan Gündüz
Computer Engineering, MSK Üniversitesi _____

Asst. Prof. Dr. Güler KALEM
Software Engineering, Atilim University _____

Assoc. Prof. Dr Beytullah YILDIZ
Software Engineering, Atilim University _____

Date: July 4, 2023

I declare and guarantee that all data, knowledge and information in this document has been obtained, processed and presented in accordance with academic rules and ethical conduct. Based on these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name : UMAIR HAIDER

Signature :

ABSTRACT

ADVERTISEMENT CLICK PREDICTION USING REINFORCEMENT LEARNING

Haider, Umair

M.S., Department of Software Engineering

Supervisor : Assoc. Prof. Dr Beytullah YILDIZ

July 2023, 63 pages

Click-through rate (CTR) prediction plays a vital role in online advertising, influencing advertisement display and advertiser cost. However, traditional methods struggle to encapsulate user preference dynamics and advertisement relevance. To address this limitation, reinforcement learning (RL) algorithms, such as Thompson Sampling, offer a promising solution by effectively balancing the exploration of new strategies with the exploitation of successful ones. In this research, we introduce a novel RL-based approach for CTR prediction which involves a custom OpenAI Gym environment to simulate real-world advertisement impressions and clicks, and an implementation of Thompson Sampling to estimate CTR dynamically, addressing the continuous evolution of user preferences and advertisement relevance. Results showed that Thompson Sampling demonstrated superior performance in CTR prediction, outperforming other RL strategies. Notably, the algorithm exhibited a confidence level nearly 10% higher than other methods. Our findings suggest that leveraging RL algorithms, particularly Thompson Sampling, can significantly enhance online advertisement selection processes, leading to higher CTRs and potentially increased revenue for publishers.

Keywords: Reinforcement Learning, OpenAI Gym Environment, Thompson Sam-

pling, Click-through-rate, Advertisement Selection

ÖZ

REKLAM TIKLAMA TAHMINI IÇIN TAKVIYELI ÖĞRENME

Haider, Umair

Yüksek Lisans, Yazılım mühendisliği

Tez Yöneticisi : Doç. Dr. Beytullah YILDIZ

Temmuz 2023, 63 sayfa

Çevrimiçi reklamcılıkta kritik öneme sahip tıklama oranı (CTR) tahmini için geleneksel yöntemler, kullanıcı tercihlerinin dinamikliği ve reklamların alakasını kapsamada zorlanırken, yeni stratejilerin keşfini başarılı olanlarla dengeli bir şekilde sağlayan Thompson Örnekleme gibi takviyeli öğrenme (RL) algoritmaları, etkili bir çözüm sunar. Bu araştırmada, gerçek dünya reklam izlenimleri ve tıklamalarını simüle etmek için özel bir OpenAI Gym ortamını ve kullanıcı tercihlerinin ve reklamların alakasının sürekli değişimini ele alan dinamik CTR'yi tahmin etmek için bir Thompson Örnekleme uygulamasını içeren yeni bir RL tabanlı yaklaşım sunuyoruz. Bulgular, Thompson Örnekleme'nin CTR tahmininde, diğer RL stratejilerinden yaklaşık %10 daha yüksek bir güven seviyesi ile, üstün bir performans sergilediğini ve bu sayede çevrimiçi reklam seçim süreçlerinin önemli ölçüde gelişebileceğini, böylece daha yüksek CTR'ler ve potansiyel olarak reklam yayıncıları için artan gelir sağlayabileceğini öne sürüyor.

Anahtar Kelimeler: Takviyeli Öğrenme, OpenAI Gym Ortamı, Thompson Örnekleme, Tıklama Orana, Reklam Seçimi

To my parents

ACKNOWLEDGMENTS

I wish to express my profound gratitude to my advisor and mentor, Dr. Beytullah Yıldız. His unwavering faith in my abilities and constant encouragement played an instrumental role in the completion of this research. His intellectual insight and mentorship are the cornerstones upon which this thesis was built.

I am also deeply indebted to my family, whose endless support during my stay in Turkey was invaluable. Their unflinching faith, moral support, and continuous encouragement, both emotionally and intellectually, were instrumental in helping me achieve my Master's degree. Their love and understanding made it possible for me to navigate the academic and cultural challenges with resilience and determination.

Lastly, I extend my sincerest appreciation to the academic community of Atilim University, Turkey. Their generosity in sharing their knowledge and wisdom, their scholarly guidance, and their nurturing academic environment enriched my Master's journey. They fostered an environment of learning that propelled my intellectual growth and made my academic pursuit at this prestigious institution a truly enriching experience.

To each one of you who contributed to this journey, I offer my deepest gratitude. Your support and guidance have made this thesis possible, and for that, I am forever grateful.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF SYMBOLS

RL	:	Reinforcement Learning
QL	:	Q-Learning
MDP	:	Markov Decision Process
DQN	:	Deep Q-Network
CTR	:	Click-Throught-Rate
DRL	:	Deep Reinforcement Learning
Ad	:	Advertisement
UCB	:	Upper Confidence Bound
CAN	:	Cross-feature Attention-mechanism Network
DIN	:	Deep Interest Network
LPC	:	Linear Predictive Coding
MiFiNN	:	Mutual Information and Feature Interaction
RTB	:	Real-time Bidding
CSV	:	Comma-separated Values
CPM	:	Cost-per-mille
CPC	:	Cost-per-click

CHAPTER 1

INTRODUCTION

The advent of digital technology has opened up a new frontier for the advertising industry, known as online advertising. It has become an essential component of a business's marketing strategy, primarily due to the rise of internet users worldwide. Online advertising presents a unique opportunity for businesses to reach a vast and diverse audience instantaneously. To maximize the effectiveness of online advertisements, companies must understand and predict user behavior accurately, which has led to extensive research in the field of Click-Through-Rate (CTR) prediction.

CTR is a widely used metric in online advertising that gauges the ratio of users who click on an advertisement to the number of total users who view the advertisement (impressions). It is a crucial performance indicator that provides insight into the success of an online advertising campaign. Accurately predicting CTR has immense practical significance for businesses since it allows them to optimize their ad content and placement, leading to increased user engagement and ultimately, higher revenue.

In 2021, display ad revenue reached approximately $238 billion[1], with global digital ad spending projected to hit $517 billion by 2023—a compound annual growth rate of 15.3% from 2020 to 2023[2]. Research indicates that the average CTR for display ads across all formats and placements is 0.35%[3], with retargeted ads achieving a higher CTR than non-retargeted ads. In the United States, the most popular advertising formats in 2021 were social media, display, and search advertising[4]. These trends underscore the importance of estimating CTR to determine the effectiveness of ad expenditures. CTR, defined as the ratio of clicks on an ad to the number of impressions it receives, serves as an indicator of an ad's attention-grabbing capacity. Factors such as ad positioning, website ranking, and display ad size can influence CTR. In

the realm of online advertising, bidding is also crucial, as businesses pay for clicks on their ads to boost CTR through platforms like Google Ads and Facebook Ads.

Machine learning has been the backbone of modern approaches to CTR prediction. Conventional machine learning techniques, such as logistic regression, gradient boosting, and deep learning models, have been used extensively to make these predictions. However, these techniques often fall short in addressing the dynamic nature of the online advertising ecosystem. Online user behavior, influenced by various factors like current trends, time of day, and personal preferences, is continuously changing, creating a highly unpredictable and non-stationary environment.

Given the limitations of traditional machine learning techniques, the need for more sophisticated approaches to CTR prediction has become apparent. This necessity has paved the way for the exploration of reinforcement learning, a subfield of machine learning that employs trial-and-error learning from interaction with an environment to optimize long-term rewards.

One promising reinforcement learning algorithm for this application is Thompson Sampling. It is a Bayesian algorithm that addresses the exploration-exploitation dilemma in reinforcement learning, a fundamental challenge where an agent must choose between exploiting the best action based on current knowledge (exploitation) and exploring new actions to gain more knowledge (exploration). The algorithm's adaptive nature makes it an appealing candidate for tackling the dynamic environment of online advertising.

In this research, we have applied the Thompson Sampling algorithm to the problem of CTR prediction in online advertising. The overarching objective was to investigate whether a reinforcement learning approach using Thompson Sampling can outperform traditional machine learning techniques in this domain. We introduce Thompson Sampling as a Bayesian RL algorithm that addresses the exploration-exploitation dilemma, a fundamental challenge in RL. The adaptive nature of Thompson Sampling makes it an appealing choice for tackling the dynamic environment of online advertising.

We have developed a custom-built environment within the OpenAI Gym platform, a

widely used framework for developing and comparing RL algorithms. This environment accurately simulates real-world ad impressions and clicks, providing a realistic setting for CTR prediction experiments.

We are using Thompson Sampling algorithm in a custom environment within our custom build OpenAI Gym. This custom environment simulates an online advertising scenario where the objective is to maximize the total reward by accurately predicting user click-through-rate. The algorithm's performance is then evaluated and compared to the performance of standard machine learning techniques. The focus is on assessing how well RL, particularly Thompson Sampling, performs in predicting CTR for online ads. We have been able to achieve almost 10% improvement in our results using our setup which proves the applied technique of using Thompson Sampling to be superior than the existing ones.

This research contributes to the rapidly growing field of machine learning in online advertising by exploring the potential of reinforcement learning, particularly Thompson Sampling, in predicting CTR. By developing a custom environment for implementing and evaluating this approach, the study provides a novel perspective on CTR prediction. We hope that our findings will inspire further research and development in this area, ultimately contributing to more effective online advertising strategies.

Our contribution to this study can be summarized as follows:

1. Proposing a novel RL-based solution, specifically through the application of the Thompson Sampling algorithm in an OpenAI Gym environment, which can provide more effective CTR predictions for online advertisements than traditional machine learning strategies.
2. Training Thompson Sampling based model in the custom built OpenAI Gym environment to achieve the better CTR predictions.
3. Conducting the same experiment for different similar algorithms for our evaluation metrics.

This thesis is organized as follows; Chapter 2 presents a literature review that discusses those researches presenting solutions for CTR prediction. Chapter 3 delves

into the background of RL, online advertisements, and OpenAI Gym which will provide the readers a chronological of the proposed solution. Chapter 4 discusses the methodology, custom OpenAI Gym environnment, and everything related to the experimental setup. The results and comparisons are presented in Chapter 5 with both descriptively and visually. Finally, Chapter 6 covers the conclusion of the work and the future scope of this research.

CHAPTER 2

RELATED WORK

The phenomenon to judge whether an advertisement will be clicked by a user when is shown a display advertisement can solved as a classification problem. Many models have been proposed by the researchers ranging from factorisation based models, logistic regression, tree models, and deep learning. We will discuss the important research done in this field in the next section.

2.1 Literature Review

A crucial aspect of CTR prediction is the feature engineering. Guo et al. [5] found a connection between the category of an application and the time period they are installed, indicating that certain features are temporally sensitive and thus significant in predicting CTR. They have found out that the restaurant delivery apps are usually in demand when it is a time for the meal. So in this scenario, it is more important to do the feature engineering on the CTR prediction datasets. It supports our approach of using reinforcement learning, as it can adapt to such temporal changes more dynamically as explained in the research [6].

Online advertising is a complex field with many factors influencing its performance. He et al. [7] highlighted the importance of identifying the correct features and using appropriate models for predicting clicks, in their case, a combination of logistic regression and decision trees. The authors have used dataset covering wide range of Facebook advertisements and suggested that the best performing strategies to predict the clicks is to logistic regression and decision trees. With the combination of

those two models, they were able to improve the system with over 3 percent which is a significant amount to be considered. They have explored that there are several factors influencing the performance of the system consisting of the correct features. Once the correct features are identified, the next most important factor is the correct models. Other factors which are not in this scope play minor role in improving the system. The right choice for the features like the data validity, fine-tuning the learning rate of the machine learning algorithm and the sampling of data can contribute to the improvement a bit but this contribution is less than the two factors like correct features and the correct model. These findings reinforce the need for reinforcement learning algorithms, which can adaptively select and optimize features based on their importance in driving user clicks [8].

Zhang et al. [9] offer a valuable perspective on how to deal with the high-dimensionality and sparsity of ad-related data. They design a hybrid model that merges the advantages of decision trees and deep neural networks. The use of decision trees allows for handling categorical variables and feature interactions, while deep learning captures intricate nonlinear patterns and deep representations. As this research involves RL for advertisement CTR prediction, exploring the integration of these methods in the RL algorithm I am using in this research could lead to more robust and accurate predictions, especially when dealing with high-dimensional data.

Wen et al. [10] focus on long-term impacts rather than immediate rewards, a concept that could be valuable in RL frameworks. They introduce an ad recommendation system that, with the aid of RL, optimizes the life-time value (LTV) of users. Furthermore, they provide a theoretical upper bound on the model's performance, offering a performance guarantee. The optimization for long-term rewards and performance assurance could contribute to a RL model by ensuring its long-term stability and reliability in predicting advertisement CTRs.

Yuan et al. [11] offers an innovative take on CTR prediction. They use deep learning to infer user intent from app icons, contributing to a better understanding of ad relevance and, ultimately, improving the CTR of mobile ads. While this work does not explicitly use RL, the concept of leveraging visual cues (app icons) to better understand user intent could be integrated into your RL model. This could be used to add another

layer of personalization, further refining the predictions and offering a unique angle for a RL-based CTR prediction research if it is used in future.

The authors of the research [12] propose a reinforcement learning-based approach to predict the CTR of an ad by incorporating contextual information such as the user's past behavior, the ad's content, and the ad's placement. They use a Deep Q-learning network to learn a policy that maximizes the expected reward, which is the CTR. The experiments were conducted using a dataset of 50,000 ad impressions, and the results showed that the proposed approach outperformed other baseline methods, including logistic regression and Random Forest. Overall, the study suggests that using a reinforcement learning-based approach can significantly improve the accuracy of ad prediction and provide better insights for advertisers.

Machine learning algorithms have increasingly been tested on large-scale datasets, demonstrating their robustness and scalability. For instance, a study by [13] showed that machine learning algorithms can successfully handle additional hurdles posed by large-scale data, corroborating our assertion that reinforcement learning algorithms, particularly the Thompson sampling algorithm, can operate effectively in online advertising scenarios with massive and diverse data which is explained in the research [14].

The authors of the research [15] have come up with the prediction rate of the CTR for the Taobao advertising dataset they have got for the research purpose. They believe that the CTR is the basic soul of the search engines, suggestion systems, and other online advertising platforms. According to them the conventional ML models like Linear Regression, Factorization and decision trees need large scale of data to find the relationships between the features and they also require a better feature engineering to supply to these models for better results. The fields like NLP, computer vision and other similar areas which use deep learning use DNN for the prediction of CTR. So, in this research they have used DIN (Deep Interest Network) with some modification for predicting the CTR. For fitting the DIN model the mixed loss is justified as the training loss. An improvement in the attention mechanism is observed as a result. Furthermore, the result show that this model performed better than the conventional deep learning models. It suggested that the DIN model can be modified to improve

the results for predicting the CTR.

In the research [16] the authors have done a research to understand how the DNN models perform in the CTR prediction. The aim is to optimize the CTR prediction for online advertising used by the businesses in shape of e-commerce websites. They have proposed a DNN based model for the prediction. Their sample consists of online advertisement links distributed on three different websites of three different businesses from the same domain of business. The DNN based models are used for predicting the clicks on these advertisements. The average prediction of 10 percent was noticed for two layered NARX network with including three extra features i.e. bounce rate, user session period and page latency. The NAR neural network without any additional external features resulted in increase of prediction to 25 percent which is close to the uncertainty of linear predictive coding (LPC).

In the study conducted by Bakhtyari and Mirzaei [17] propose an innovative approach of employing boosting models, specifically emphasizing the importance of meticulous feature engineering and parameter tuning. They argued that the optimal configuration of the XGBoost model could yield significant performance enhancements. Their methodology involved tweaking the features and eliminating redundant data to establish the effectiveness of their proposition. Subsequently, they applied a grid search scheme to select the most suitable hyper-parameters for their model. The research utilized the Avazu dataset, encompassing 11 days of user interactions exceeding 40 million, each with more than 24 attributes. The data was divided into three subsets for experimentation. The results obtained affirmed their hypothesis, indicating that the proposed XGBoost model, augmented with careful feature engineering and hyper-parameter tuning, delivered superior performance.

In their research [18], Wang, Dong, and Han introduce an approach called Mutual Information and Feature Interaction (MiFiNN). This method synergistically blends feature interaction and mutual information to determine the weight of each feature. The process involves feature combination, extraction of feature interactions, and subsequent incorporation of these newly engineered features into a deep neural network (DNN) model.

The research paper [19] delves into real-time bidding (RTB), a contemporary per-

impression strategy in online advertising that emphasizes real-world dynamics and user interaction patterns. In response to the dynamic nature of data and limited features accessible from user interactions in real-time scenarios, the authors innovatively utilized a Deep Neural Network (DNN) model, augmented with additional data sources like user applications. This thoughtful integration, alongside layering with user-ad interactions, creates a robust model capable of efficiently predicting CTRs. The resulting model not only illustrates the practical effectiveness of this approach but also aligns with our research in Thompson Sampling reinforcement learning, given its focus on dynamic user engagement and interaction patterns.

The researchers in their article, [20] have used a model based on Federated Factorization Machine (FedDeepFM). This model implements the CTR prediction without exposing the user private data. The model is actually updated by feeding gradient information of the user to the distributed factorization machine on user level. The loss is usually caused by heterogeneous data from the user, to deal with this problem they have come up with constructing a cluster of federation learning for the factorization machine which they call FedDeepFM. After the implementation of the prototype the results show there is 8 percent improvement than the conventional factorization machine.

CHAPTER 3

BACKGROUND

The rapid advancements in the field of reinforcement learning (RL) and deep neural networks (DNNs) have significantly impacted various sectors, including online advertising and digital marketing. The concept of training DNNs directly from raw input, using incremental adjustments based on stochastic gradient descent, has demonstrated its potential in real-world applications [21].

In the realm of RL, this approach was leveraged successfully in training algorithms to master Atari games [22]. The researchers connected an RL algorithm to a DNN, which processed RGB images and trained the data using stochastic gradient updates. They applied a method called experience replay where the agent's experiences at each step were stored into a replay memory and subsequently used for mini-batch updates.

Reinforcement learning problems can be represented as Markov Decision Processes (MDPs) or Partially Observable Markov Decision Processes (POMDPs) [23]. These problems can be further divided into three main categories: prediction or policy evaluation problems, control problems, and planning problems. Prediction problems involve calculating the value of a state or action for a given policy, control problems aim to determine the optimal policy, and planning problems focus on the creation of a value function or policy using a model.

The RL methods used to solve these problems can be value function or policy-based, on-policy or off-policy based, and can either involve function approximation or not. In instances where there is a large or continuous space, function approximation is used to determine an approximation of the entire function using examples of that function.

OpenAI Gym [24], a research toolkit for RL, offers a collection of environments based on POMDPs. During interaction with the environment, an agent takes an action and receives an observation and reward at each step. In RL, the environment has been formalized as a POMDP [23].

In this context, the principal aim of reinforcement learning algorithms is to maximize the total reward that an agent receives while interacting with the environment. An agent's experience is divided into a series of episodes, each beginning with an agent's state selected randomly from a distribution. Interaction continues until the environment's terminal state is reached. The primary goal of this episodic RL is to maximize reward in each episode, achieving a high-performance level in as few episodes as possible.

The foundation for understanding the essence of this thesis is laid in this chapter. Section 3.1 presents the fundamentals of Reinforcement Learning, focusing on key algorithms like the Markov Decision Process, Monte Carlo, Multi-armed Bandits, and Temporal Difference methods. Additionally, this section discusses Deep Reinforcement Learning algorithms like DQN as part of the discussion on value-based and policy gradient-based methods. Section 3.2 elucidates the dynamics of online advertisement networks and the entities involved, providing insight into the workings of online advertisements. Lastly, section 3.3 delves into the OpenAI Gym environment and its components, illustrating how it can be customized to suit online advertisements. Further details on customizing the gym to fit the needs of online advertisements will be discussed in Chapter 4.

3.1 Reinforcement Learning

Reinforcement learning is a concept widely used in machine learning and artificial intelligence. It involves trial-and-error behavior, where the agent interacts with the environment provided and experiences are stored to influence the upcoming actions that the agent will take. The agent learns from its previous experiences and aims to maximize its reward or minimize its cost. This approach is often used in scenarios where the optimal solution is not known or where the environment is constantly

changing.

At each time step the agent takes an action from an available set of actions and changes its state to an available set of states. The action taken by the agent changes the environment, and the agent enters a new state. The agent then receives a reward from the environment based on the action it has taken, which serves as feedback to indicate whether the action taken was good or not. The agent uses this feedback to adjust its future actions to achieve a higher reward.

Understanding the basic concepts of Reinforcement Learning is crucial for comprehending the advanced concepts of Reinforcement Learning discussed. In the subsequent sections we cover the various topics related to RL such as Markov Decision Processes, Deep Reinforcement Learning, Monte Carlo Methods, and Temporal-Difference Learning, in a more chronological way to understand where exactly the approach which is Thompson Sampling falls under the subcategories of RL.

3.1.1 Model-based methods

3.1.1.1 Dynamic Programming

Dynamic programming is a method for solving sequential decision-making problems that involve uncertainty, such as those found in reinforcement learning. The approach involves breaking down a complex problem into simpler sub-problems and solving them recursively. This is achieved by using a mathematical technique called Bellman's equation, which expresses the value of a state or action as a function of the expected immediate reward and the value of the next state or action. The equation takes the form of a recursive relationship that can be solved using iterative methods. By computing the optimal value function or policy, dynamic programming allows for efficient and effective decision-making in a variety of domains.

3.1.1.2 Markov Decision Process

The Markov Decision Process (MDP) is a mathematical framework that forms the basis of many reinforcement learning problems. The key characteristic of MDP is the

Markov assumption, which suggests that the future states of a system only depend on the current state, not on the sequence of events that preceded it. This assumption simplifies the complexity of the problems by ensuring that the previous actions, a(1), a(2), ... a(t-1); where (a) is an action, undertaken by the agent do not influence the current state and the behavior of the environment at the time step t. A Markov Decision Process is defined by the tuple (S, A, P, R, Y); where (S) is a set of states, (A) is a set of actions, (P) is transition probabilities, (R) is reward function and (Y) is discount factor. An MDP is finite if both the set of states (S) and the set of actions (A) are finite.

Discounted Expected Reward The ultimate objective of reinforcement learning is to maximize the long-term reward rather than immediate returns. The concept of the discounted expected reward aids in accomplishing this by adjusting the focus from immediate gains to future rewards.

Consider an environment with six rooms, where the agent's goal is to reach room five (refer to Figure 3.1). Each room can be thought of as a state in the Markov Decision Process, and moving between rooms corresponds to taking actions. The immediate return for the agent corresponds to the negative of the agent's current distance to room five. This represents the cost of taking each action and is incorporated into the calculation of the expected reward.

If we concentrate solely on immediate returns, the agent might select room three because it offers a higher immediate reward than rooms one or two. This could be because room three is physically closer to room five, so the negative distance - and therefore the immediate reward - is higher. However, once in room three, the agent cannot reach the target (room five) due to some constraints in the environment, such as a wall or a locked door, and is essentially trapped. This demonstrates how focusing exclusively on immediate rewards can lead to suboptimal long-term outcomes.

In contrast, when the agent aims to maximize long-term reward, it would initially accept lower immediate rewards by moving through rooms one, two, and so forth, ultimately reaching the target room and maximizing the sum of all rewards. This is where the discounted expected reward concept shines, as it allows the agent to balance

the trade-off between immediate and future rewards, and make decisions that lead to the greatest total reward over time.

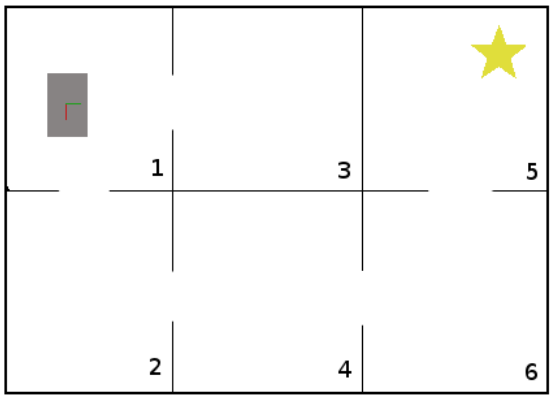

Figure 3.1: Immediate reward vs. expected return

The discounted expected return, represented by the cumulative sum of all future rewards, takes into account the effect of future rewards. The discount factor determines how much weight is given to these future rewards. A discount factor of 1 means future rewards are weighted equally to immediate ones, whereas a discount factor of 0 implies that only the immediate reward is considered.

MDP tasks can be episodic or continuous:

- **Episodic:** In this scenario, the training process is divided into episodes. Each episode concludes when a terminal state is reached, after which the environment is reset, and a new episode commences. The terminal state yields a return of 0 and is achieved within a finite number of time steps.

- **Continuous:** For ongoing problems that can't be partitioned into separate episodes, the task is considered continuous. Consequently, the time steps to reach a terminal state, denoted by T, are infinite.

3.1.1.3 Monte Carlo Method

The Monte Carlo approach deal with solving reinforcement learning problems based on episodic tasks, where we don't know the model of the environment. The method consists of repetitive iterations and converges with the growing number of episodes toward the optimal policy.

The Q-table in the Monte Carlo method stores action-value pairs for each possible state-action combination. The value recorded for each pair is essentially the mean of all returns observed across numerous episodes. Each time an agent experiences a state-action pair, the corresponding entry in the Q-table is updated.

The updated equation for Monte Carlo can be represented as follows:

$$Q(S_t, A_t) = Q(S_t, A_t) + \frac{1}{N(S_t, A_t)} (G_t - Q(S_t, A_t)) \tag{3.1}$$

In this equation 3.1, $Q(S_t, A_t)$ represents the current action-value estimate for a state-action pair (S_t, A_t). The term $N(S_t, A_t)$ denotes the number of times the state-action pair (S_t, A_t) has been encountered. The quantity G_t is the return received after time-step t in the episode. The equation essentially adjusts the current action-value estimate towards the actual return G_t, at a learning rate of $1/N(S_t, A_t)$.

The Monte Carlo method proceeds iteratively through each episode. During the Policy Evaluation step, the agent employs a policy, denoted by π, to navigate through an episode. This policy dictates the probability of selecting each possible action in a given state. The aim is to iteratively refine this policy based on the learning gained from successive episodes to maximize the expected return.

3.1.2 Model-free methods

3.1.2.1 Temporal Difference Methods

Temporal Difference (TD) techniques are widely used in reinforcement learning due to their advantages over other methods. One such advantage is that TD techniques are

suitable for online learning and continuous RL tasks, where the agent interacts with the environment in real-time. The policy is updated at each time step t, allowing the agent to learn and adapt quickly without needing to wait for completed episodes.

Another advantage of TD techniques is that they do not require a model of the environment. This means that the agent can learn and improve its policy directly from the experiences it encounters, without needing to understand the underlying dynamics of the environment.

Experiments have shown that TD approaches often tend to converge more quickly than the Monte Carlo method. This is because TD methods use an iterative approach to update the value function, while Monte Carlo methods require a complete episode to be completed before any updates can be made.

In TD-methods, the real anticipated return G_t is not available and is estimated using the TD-target. The TD-target can also be viewed as an approximation of the Bellman equation 2.23. The TD-target is calculated by taking the sum of the immediate return and the discounted anticipated value of the next state: $R_{t+1} + \gamma V(S_{t+1})$. The discount factor γ is used to weigh the importance of future rewards, and the value function V is used to estimate the long-term value of each state.

Overall, TD techniques are a powerful tool for reinforcement learning and have been used in many real-world applications. By updating the policy at each time step and using the TD-target to estimate the expected return, the agent can quickly learn and adapt to changing environments without needing a complete model of the environment.

Sarsa Sarsa, an acronym representing State-Action-Reward-State-Action, operates as an on-policy Temporal Difference (TD) control approach. In each iteration, Sarsa uses a five-tuple $(St, At, Rt+1, St+1, At+1)$ to update the Q-value, denoted as $Q(s, a)$, of the state-action pair (St, At) within the Q-table.

Initially, the agent performs action At in state St, subsequently receiving an immediate reward, $Rt + 1$. Thereafter, policy π determines the ensuing action $At + 1$ that will be executed in the next state, $St + 1$. The TD-target is the summation of the immediate

reward, $Rt+1$, and the Q-value of the upcoming state-action pair $(St+1, At+1)$. This TD-target approximates the true expected return Gt.

The Q-value is updated according to the equation provided below, also known as the action-value update policy:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \right]. \tag{3.2}$$

In this equation, α is the learning rate dictating the step size for each update, while γ is the discount factor determining the present value of future rewards. The term $R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})$ represents the estimated return following the current policy, and $Q(S_t, A_t)$ is the current estimate. The difference between these two values forms the TD error. The Q-value is adjusted by a fraction α of this TD error, resulting in a new estimate that is incrementally closer to the expected return.

Q-Learning Q-Learning is a renowned off-policy Temporal Difference (TD) control method. In this context, 'off-policy' signifies that the policy π does not play a role in updating the Q-table. Q-Learning contrasts with Sarsa in that it doesn't use the action-value of the next state-action pair, $Q(St+1, At+1)$. Instead, it incorporates the maximum Q-value of the next state, $St+1$.

The action-value update mechanism of Q-Learning is encapsulated in equation 3.3.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right] \tag{3.3}$$

In this equation, α denotes the learning rate, guiding the magnitude of each update, and γ is the discount factor that weighs the importance of future rewards. The term $R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$ represents the estimated return if the best available action is chosen in the next state, and $Q(S_t, A_t)$ is the current Q-value estimate. The difference between these two values generates the TD error. The Q-value is then updated by a fraction, α, of this TD error, yielding a new estimate that aligns more closely with the expected return.

3.1.2.2 Deep Reinforcement Learning

In the previous section, we have discussed the RL. The usage of a table limits the classical approaches to tasks with a reduced variety of states and actions. In real-world problems, the state space can quickly get big. It is not possible to visit all feasible states to fetch the value for all action-state pairs. Moreover, the dimension of the table is restricted as a result of memory constraints in hardware. On the other hand, knowledge regarding similar states is not shared. This might lead to much better representation and lower training times.

To overcome the mentioned restrictions, a common method is to replace the value table with a Deep Neural Network as a function approximator. Their capability to approximate non-linear functions and to pull out essential features from raw inputs makes it possible to generalize over unseen states.

Value Based Methods Value-Based Methods improve Temporal Difference Methods from classical Reinforcement Learning discussed previously. The suggestion is to replace the one-to-one value table and approximate it with a Deep Neural Network. The network's output delivers probabilities for each possible action. A traditional policy lays on top of the network output to select the final action (e.g., epsilon-greedy policy).

Deep Q-Network (DQN) Combining Q-Learning with non-linear function approximation has been explored in the past and also did not lead to great success because of unstable learning. In 2015, the DeepMind group [22] presented a strategy– called deep Q-Network (DQN)– that showed great success. They merged the model-free, off-policy Q-Learning with. Deep Neural Networks. As input data, high-dimensional raw sensory input with no previously hand-crafted features are utilized. This end-to-end architecture enables the network to extract relevant features by itself. The outcome of the Q-network offers a probability distribution over all feasible discrete actions. This enables one to establish the best action for a given state with a single forward pass. Especially, the problem of unpredictable learning has actually been enhanced with two added mechanisms, called experienced replay and also icy target

network, which will certainly be further described in the following.

The concept of experienced replay is to store the agent's experiences St, At, Rt +1, St +1 in a buffer that can hold the number of experiences in total amount. In each training step, a batch of experiences is evenly sampled from the buffer and fed to the network. Experienced replay eliminates the correlations in the data series as well as feeds the network with independent data. It also makes certain that old experiences are duplicated every now and then. It has a smoothing result over changes in the data circulation.

In DQN the network is updated according to the loss feature from equation 3.4. The loss function is calculated by taking the squared TD-error.

$$L_i(\theta_i) = \hat{\mathbb{E}}_t\left[\left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a, \theta_i^-) - Q(S_t, A_t, \theta_i)\right)^2\right] \tag{3.4}$$

In equation 3.4, the second mechanism frozen target network is presented too. Two networks with the exact same framework, but different weights are used: θ for the Q-network and also θ^- for the target network. The Q-network is frequently updated according to the loss function from equation 3.4, while the target network is upgraded by copying the parameters of the Q-network to the target network $\theta^- = \theta$ every C time step. Therefore, the weights of the target network θ^- are held frozen for C time steps. It smooths oscillating policies as well as causes extra-supported discovery.

Improvements in DQN After the publication of the effective DQN method, it has been investigated extensively as well as a number of publications with improvements followed. Rainbow [25] analyzes all related enhancements with the initial DQN technique and also applies a combination of all improvements called Rainbow DQN. In the adhering to paragraphs, three major improvements are briefly and also intuitively introduced.

Double DQN The Double DQN approach [26] seeks to address the overestimation of Q-values, a common issue, particularly in the initial stages of the learning process. During this phase, it's likely that incorrect actions are assigned the highest Q-values.

Double DQN introduces a strategy of utilizing two different Q-networks, denoted as θ and θ^{-}, for approximating the Temporal Difference (TD) target. This results in a more robust learning process. Given that the DQN algorithm already incorporates two separate networks, Double DQN seamlessly integrates into the framework by slightly modifying the loss function as shown in equation 3.5.

$$L_i(\theta_i) = \hat{\mathbb{E}}_t\left[\left(R_{t+1} + \gamma Q\left(S_{t+1}, \underset{a}{\operatorname{argmax}} Q(S_{t+1}, a, \theta_i), \theta_i^{-}\right) - Q(S_t, A_t, \theta_i)\right)^2\right] \tag{3.5}$$

In this equation, θ_i and θ_i^{-} are parameters of the online network and target network respectively. Here, the maximum action is selected using the online network θ_i, but the Q-value of this action is evaluated by the target network θ_i^{-}. This decouples the action selection from the Q-value estimation, thus curbing the propensity for overestimation.

Policy Gradient Methods Policy Gradient Methods optimize the policy $\pi(a \mid s, \theta)$ straight as opposed to learning a value function as well as selecting the activities based on it (e.g. e-greedy policy). The quality of each policy can be determined by the policy's efficiency procedure J(θ). The objective function of Policy Gradient Methods in equation 2.31 optimizes the scalar value J(θ).

$$\theta^{*} = \underset{\theta}{\operatorname{argmax}} J(\theta) \tag{3.6}$$

The policy's parameter θ is updated through gradient ascent. Gradient ascent is the reverse of gradient descent as well as updates the parameters θ_t in the positive direction of the gradient of the policy's performance measure $\nabla_\theta J(\theta)$ (see equation 2.32). Moreover, the learning rate α specifies, how strongly one steps in the direction of the gradient.

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\theta_t) \tag{3.7}$$

One positive aspect of Policy Gradient Methods is their consistent convergence property due to the fact that the policy is updated directly and also hence boosts efficiency

at each time step. Value-based methods update the value function at each time step. A small change in the value function can cause a drastic change in the policy output. Therefore, value-based techniques typically deal with huge oscillations during training. Specifically, Policy Gradient Methods can handle boundless and also continuous action spaces. As opposed to determining a Q-value for each possible discrete action, the action can be estimated directly, e.g. the speed of the mobile robot is approximated directly by the agent. The third positive aspect of Policy Gradient Methods is their ability to discover stochastic policies, i.e., actions are picked with a particular likelihood. It is particularly essential for uncertain, partially observable environments. The huge disadvantage of Policy Gradient Methods is that they instead merge to a local optimum than to the global optimum. [8]

Actor-Critic Architecture A Policy Gradient Method that takes advantage of the value function v(s) to learn the policy parameters θ is called Actor-Critic Architecture. Figure 2.10 illustrates the basic idea of the style: The Actor represents the present policy as well as produces an action for an offered input state s. The Critic represents the value function v(s) as well as calculates the expected value for a provided input state. A usual method is to update both networks with the TD-Error, talked about previously. The anticipated values of the current and also the following state that contributes to the TD-Error are approximated with the Critic. Thus the Critic's outcome adds to the Actor's update essentially.

3.1.2.3 Multi-Armed Bandits

Multi-armed bandit problems are a class of sequential decision-making problems in which an agent must choose between multiple options, each with an unknown probability distribution of rewards. The goal of the agent is to maximize the total reward over time, while simultaneously learning more about the reward distributions associated with each option. This problem has been widely studied in statistics, operations research, and machine learning, and has numerous applications in areas such as clinical trials, online advertising, and recommendation systems.

The multi-armed bandit problem is an important and challenging problem in rein-

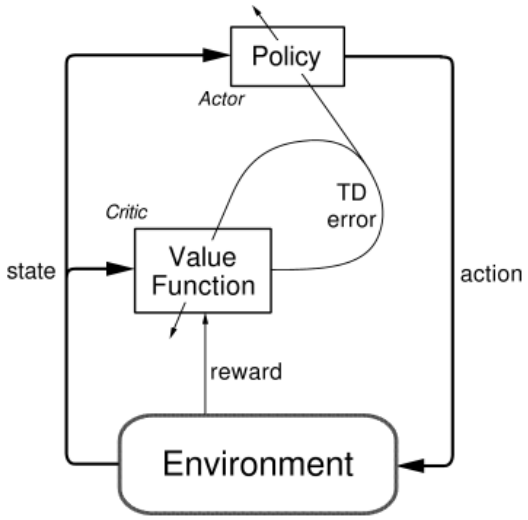

Figure 3.2: Actor-Critic: The actor works on a policy function while the critic works on a value function. Both networks get updated based on TD-Error

forcement learning, a subfield of machine learning that is concerned with how agents can take actions in an environment to maximize a cumulative reward signal. In this problem, the agent must balance exploration (trying out different options to learn more about their reward distributions) and exploitation (choosing the option that is currently believed to be the most rewarding).

There are several algorithms that have been proposed to solve the multi-armed bandit problem, each with its own strengths and weaknesses. In this article, we will discuss some of the most popular algorithms, including Upper Confidence Bound (UCB) [27], Thompson Sampling [28], Epsilon Greedy algorithm [29], and Softmax algorithm [8].

Bayesian Bandit Algorithms Bayesian Bandit Algorithms are a subset of multi-armed bandit algorithms that incorporate Bayesian statistics into their decision-making process. These algorithms use prior probabilities and Bayesian updating to make decisions about which ”arm” to choose, rather than relying solely on the empirical reward rates. One of the major advantages of Bayesian Bandit Algorithms is that they naturally balance exploration and exploitation through probabilistic decision-making.

Thompson Sampling Thompson Sampling is another popular algorithm for the multi-armed bandit problem. It works by sampling from the posterior distribution of the reward distribution associated with each option, and then choosing the option with the highest expected reward. Thompson Sampling has been shown to have good performance in many settings, and is particularly well-suited to cases where the reward distributions are complex or have multiple peaks.

The implementation of the Thompson Sampling algorithm follows the exact same strategy initially, just like Upper Confidence Bound (UCB) as:

Thompson Sampling value = exploitation + exploration

where the exploitation is the current estimated value of the advertisement, but the exploration here is a sample drawn from the posterior distribution over the advertisement's value. The algorithm selects the advertisement with the maximum Thompson Sampling value to play next as it does in Upper Confidence Bound (UCB) algorithm.

Upper Confidence Bound (UCB) The UCB algorithm is a popular approach to solving the multi-armed bandit problem. It works by choosing the option with the highest upper confidence bound, which is a measure of the uncertainty of the reward distribution associated with that option. The UCB algorithm has been shown to have good performance in many settings, including cases where the reward distributions are non-stationary or where there are many options. The UCB agent we have under consideration is an implementation of the UCB algorithm for ads click prediction. The algorithm updates the value for the action of the previous act() call based on the reward received, and tests each ad once. Then, it computes the UCB values for each ad using the formula:

UCB value = exploitation + exploration

where exploitation is the current estimated value of the ad, and exploration is a measure of the uncertainty of the ad's value. The algorithm selects the ad with the maximum UCB value to play next.

Epsilon Greedy algorithm The Epsilon Greedy algorithm is a simple but effective approach to solving the multi-armed bandit problem. It works by choosing the option with the highest estimated reward with probability (1-epsilon), and exploring a random option with probability epsilon. The Epsilon Greedy algorithm is easy to implement and has been shown to have good performance in many settings, but can struggle in cases where the optimal option is rare or hard to find.

The Epsilon Greedy algorithm for ads click prediction chooses the ad to play next based on the value of epsilon. If a random number is less than epsilon, the algorithm chooses an arm randomly, otherwise, it chooses the arm with the highest estimated reward so far.

Softmax algorithm The Softmax algorithm is commonly used in selecting among a set of options with probabilities proportional to their values [30]. In the context of reinforcement learning, the Softmax algorithm is often used to balance exploration and exploitation in multi-armed bandit problems [8]. The Softmax algorithm computes the probabilities of selecting each available action at a given state based on their expected values, which are computed using a temperature parameter.

The Softmax algorithm we are using for ads click prediction computes the temperature based on the remaining time and the beta parameter, and then computes the probabilities of choosing each arm using a softmax function. The algorithm chooses the arm to play next using weighted random selection based on these probabilities.

3.2 Online Advertising

We have different entities involved while conducting this research. We need to understand them first to understand the whole structure of how ads work and what is their revenue model. Online advertising refers to the use of the internet to deliver promotional marketing messages to consumers. There are several different models for delivering online advertising, but a common one involves three main parties: publishers, ad networks, and advertisers as shown in Figure 3.3.

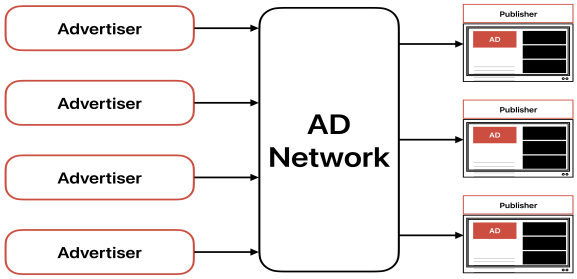

Figure 3.3: Ad Network Diagram

3.2.1 Ad (Advertisement)

First of all, we will try to define the term Ad (Advertisement). We are only taking online online advertisements into consideration. An advertisement is a form of marketing communication that aims to promote a product, service, or idea to a target audience. In the context of an online ad network, publisher, and advertiser scenario, an advertisement refers to a message or content that is designed to promote a product, service, or idea to a specific group of people. In this scenario, the advertiser is the company or individual who is promoting the product, service, or idea, and the ad network is a platform that connects advertisers with publishers (websites or apps) who are willing to display the advertiser's ads to their audience. The publisher is the website or app that displays the advertiser's ads to its audience in exchange for a fee. Advertising can take many forms, such as banner ads, display ads, video ads, native ads, and more. The goal of advertising is to reach a specific group of people with the message and persuade them to take some kind of action, such as purchasing a product, signing up for a service, or visiting a website.

3.2.2 Advertiser

This entity can be a company or an individual who wants to put out an ad for a certain purpose on the internet. The first step they take is to find an online ad network like Google Adsense and try to advertise their products or services on their platform. Advertisers are businesses or organizations that want to advertise their products or services to consumers. They may advertise directly on a publisher's website or through

an ad network to reach a larger audience. Advertisers typically pay for ads based on either the number of times an ad is clicked on (CPC—cost-per-click) or the number of times it is viewed (CPM—cost-per- impression). This means they pay a specific amount every time someone clicks on or views the ad.

3.2.3 Ad Network

This can be a company or a service provided by a company and this network serves as a bridge to connect advertisers and publishers. A good example of this can be the Google Adsense program. Ad networks serve as intermediaries that link advertisers to publishers. They often work with many publishers, giving advertisers the opportunity to reach a broad audience through a single platform. Ad networks may also focus on specific forms of advertising, such as display, search, or video ads. They earn money by charging advertisers for access to their platform and taking a percentage of the ad revenue generated by the publisher.

3.2.4 Publisher

This is the entity that offers space on its digital platform. This platform can be a website or a mobile application. Publishers are individuals or organizations that own websites or other online properties that show advertisements. This could be an individual with a blog or social media account, or a company with a network of websites. Publishers earn money by displaying ads on their properties and receiving a commission or other payment for each ad impression or click. In the online advertising model, publishers and ad networks collaborate to show ads to consumers and make money from them. Advertisers pay for the opportunity to reach these consumers through the ad network or publisher's platform. This model enables the creation and distribution of a variety of advertising content to a large audience, helping businesses to reach potential customers and promote their products and services.

3.2.5 Click-through-Rate (CTR)

Click-through rate (CTR) is a key metric in the online advertising industry, as it measures the effectiveness of an ad in terms of how many people actually click on it. CTR is typically expressed as a percentage, calculated by dividing the number of clicks on an ad by the number of impressions (times the ad was displayed). A high CTR is generally seen as a sign that an ad is effective and engaging, while a low CTR may indicate that the ad is not resonating with the target audience. In the context of online advertising, CTR is relevant for all three parties involved: publishers, ad networks, and advertisers. For publishers, CTR is important because it determines the revenue they can generate from an ad. Publishers typically earn money from ads by charging advertisers on a cost-per-click (CPC) or cost-per-impression (CPM) basis, meaning they receive a certain amount each time someone clicks on an ad or views it. A higher CTR means more clicks and more revenue for the publisher. For ad networks, CTR is also important because it determines the revenue they can generate from an ad. Ad networks typically earn money by charging advertisers a fee for access to their platform and taking a cut of the ad revenue generated by the publisher. A higher CTR means more clicks and more revenue for the ad network. For advertisers, CTR is important because it determines the effectiveness of their ad campaign. Advertisers typically pay for ads based on either the number of times an ad is clicked on (CPC) or the number of times it is viewed (CPM). A higher CTR means that more people are clicking on the ad, which may indicate that the ad is resonating with the target audience and achieving its desired effect. A low CTR may indicate that the ad is not effectively reaching or engaging its target audience, and the advertiser may need to adjust their ad strategy to improve its performance.

3.2.6 Applying RL in Online Advertisement

The click-through rate (CTR) of an online ad is the proportion of users who click on the ad after it is shown to them. The CTR of an ad can be connected to the multi-armed bandit problem, a type of problem in reinforcement learning that involves balancing exploration (trying different options) and exploitation (choosing the option with the highest expected reward). In the multi-armed bandit problem, there are several

”arms,” or options, each with a probability of yielding a reward. The goal is to maximize the total reward by choosing the best arm. In online advertising, the arms could represent different ads and the reward could be the CTR of the ad. To maximize the CTR, the advertiser must balance exploration and exploitation. If they only display the ad with the highest known CTR, they risk missing out on higher CTRs that could be achieved by trying out other ads. On the other hand, if they only explore and try out new ads, they may not maximize their CTR because they are not exploiting ads that have already proven to be effective. The multi-armed bandit problem can be solved using reinforcement learning algorithms, which learn from past experience to make decisions about which arm to choose to maximize the reward. In online advertising, these algorithms can be used to optimize the selection of ads to display to maximize the CTR.

3.3 Gym Environment

There are two basic parts of any type of RL problem. First is the agent, which is the algorithm execution. Second, is the environment the agent interacts with [31]. The actions of an RL agent are not configured explicitly, but implicitly, by thoroughly creating the environments in which they are trained [31]. The RL algorithm itself is typically very general, and only imposes limitations on problem information such as whether the action space and observation space are discrete or continuous [24].

We can say that the algorithm application dictates how the agent learns, but the environment determines what the agent learns [31]. Specifically, the implementation of the system simulation and the reward function shape the resulting agent policy [31].

However, in the case of predicting the click-through rate (CTR) problem, there are three fundamental problems that must be addressed [32].

3.3.1 Exploration vs exploitation problem

This problem points out the natural dilemma between exploration and exploitation and how much trade-off can be done between them while trying out things. The first one

suggests how the whole space can be traversed to understand the scope of the whole search space. The last one is more about how the current already explored space can be refined which means that the area of search space is not supposed to be extended [33].

For example, you always go to a restaurant to eat your favorite food. You are very much familiar with the taste of the food which is being served by this restaurant. One day, you found a place that has the same menu as the other restaurant. The decision here is on you, whether you stay with the old restaurant which is more familiar to you (exploitation) or you go to new places to find out different tastes (exploration) to come to a conclusion that which place is better [33].

3.3.2 Delayed reward problem

This problem points to the long-term reward that the agent learns after its interaction with the environment with the hit and trial method. It is not just the immediate reward that matters, but the long-term reward [8]. For example, a toy car with machine learning running in a room with hurdles learns them by hitting them. Eventually, it learns all the hurdles and knows what the paths lead to them and in the end, it drives without hitting any hurdles in the room. So, the ultimate goal of reinforcement learning is to learn all the sequences leading to success by observing all the rewards in each state [33].

3.3.3 Generalization problem

This problem is related to the last problem we discussed, which is the hit and trial method. In that situation, we are talking about exploring every state which is leading to reward or not-reward. So, in the example where we discussed exploring the restaurants if we have 5 million restaurants instead of 3 or 4 then you can realize how complicated the exploration can get. Then we will be thinking that exploring all of them will be really hard, then we have to think about the generalization of the scenario by defining certain criteria like a restaurant in a certain area that has the best-fried chicken. So, in generalizing, the restaurant may result in not visiting some of

them or even finding out hidden ones [33].

In this thesis, we will limit our scope to the exploration-exploitation problem. One of the problems which comes under this domain is the multi-arm bandit problem [34].

3.3.4 Fitting Multi-arm bandit in Gym Environment

This problem explains the exploration-exploitation dilemma very well. Suppose you are at a casino with many slot machines in front of you and the probability of winning a reward for one chance at each machine is unknown to you. Now, what will be the best technique to secure the highest long-term reward? These slot machines are called ”bandits” because in most cases people lose all of their money at these machines[34].

A simple approach will be trying again and again on a single machine until the reward turns ”true,” which is not good and can be exhausting and wasteful to reach the highest long-term reward [33].

Let’s suppose we have a website that consists of multiple articles, and we have to decide which article we have to show to the visitors. We have no idea about the visitors, and we are also not sure which link will be clicked by the visitors. Now, we have to decide in which order the links should appear to the visitors so that the interaction of visitors with the articles increases. But since we have no knowledge about the visitors, it is difficult to decide the strategy to show the articles [33].

Another example we can take of a pharmacy company, which intends to give the best medicine to a bunch of patients. Now, the company has to decide who could be the best fit for a certain medicine, also keeping in mind that the medicine should not be given to the wrong patient so this chance should be minimized [33]. If we talk about Bernoulli’s multi-armed bandit problem it can be defined as a tuple of (A, R), which describes as:

- There is K number of machines with the probability of rewards $\theta 1, \ldots, \theta K$.
- The set of actions A represents the interaction with each machine. The expected value of the action is a reward which can be described as $Q(a)=E[r—a]=\theta$. The time step is defined as t in this equation as $Q(at)=\theta i$.

- The R in this tuple is the reward function, and if we consider Bernoulli multi-arm bandit problem we have a reward r at the time t. The equation for this may be rt=(at)rt=R(at) which can be either reward or not-reward.

So, eventually, when we look at the multi-arm bandit problem we have three strategies in our hands.

1. We select a machine randomly without any specific criteria and try to get the reward. This is a bad approach to reaching any long-term reward.
2. We try to explore randomly but when we get some rewards then we try to exploit them to get some better rewards.
3. We look smartly through the exploration without depending on a fixed probability for each reward, which means the calculation is smart in exploring without exploiting the rewards again.

3.3.5 Observations

With each interaction with an environment, the agent gets an observation that defines the existing state of the environment [8]. The agent's behavior is dictated by a policy that maps these states to actions, so what details are made available to the agent via the observation is of vital importance for its efficiency. By giving useful state information to the system, the formula can learn much better policies more quicker [31]. Nonetheless, one of the reasons to use deep neural networks as feature approximators is that they have the ability to learn useful information from the input on their own, and one of the objectives of DRL in general is to make agents which can gain from raw input information, without the demand for customizing features [31].

The environment observations are limited to standard state values and errors [33].

3.3.6 Rewards

The DRL agent makes use of a return mechanism which is generally a reward or a punishment strategy to get feedback about its own performance [8]. The policy is

enhanced to maximize the collected returns from this function, which suggests that the design of the reward function essentially dictates the behavior of the agent [8]. This can be made use of both to reward desirable behavior, as well as to punish undesirable behavior. Standard control objectives such as minimizing or restricting the rise time and overshoot, rejecting disturbance, and constricting input and input adjustment can be achieved by including relevant terms in the reward function [8]. Rise time can be minimized, for example, by using the error, overshoot can be limited by punishing the state value being higher than the set-point, and input and input change can be added as costs directly to achieve preferential behavior [8]. The standard reward function in the created environments is simple.

The environments are relatively straightforward, so the reward functions can likewise be both simple and basic. In more multi-dimensional systems or environments, a well-formulated reward function is crucial to the learning process and to finding a good policy [8].

3.3.7 Interface

The environments are implemented using the OpenAI Gym [24] interface. In Gym, the core interface is implemented as a class called Env, which encapsulates an environment, which can be either partly or totally visible, concealing the specifics of the environment characteristics. Various environments are implemented by inheriting the Env class, and also overriding its main methods and also attributes. The main API methods are:

- **reset():** This function resets the state of the environment and returns to the initial state. This is used to initialize the environment and also used to reinitialize it.
- **render():** This function is used to visualize the progress and the performance of the agent. In our case there are two rendering modes available, one is the ”human” which is a graphical form and the other is the JSON array.
- **step(action):** This method takes action as a parameter and runs for one-time step and then returns the next time step.

- **close():** This function is used to clean up the environment.

For example, a simple step() call will have this line of code:

$observation, reward, done, info = env.step(action)$

In the above line, the *env* is an instance of environment class and the *action* variable is the action to be performed by the agent.

Both reset() and step() function return the following variables:

- **observation:** This is an observation object which consists of a new environment state for the agent's current observation.
- **reward:** This variable is basically the reward value for the agent in return of the action it performed.
- **done:** This is a conditional variable to specify the completion of the episode. When it reaches the "true" condition, the step() function calls from that point returns undefined results.
- **info:** This variable contains the debugging information of the environment to exploit the environment in more detail.

The *Env* class further contains three very crucial attributes:

- **observation space:** This variable defines the observation space for the valid state inside the environment.
- **action space:** This provides an action space for the agent to choose valid actions from.
- **reward range:** This attribute sets a boundary for possible rewards for the agent actions.

The action space and observation space attributes are objects of type Space, which is implemented by the Gym framework, especially for action and observation spaces [24]. The most common spaces are Discrete and Box. The Discrete space consists of

a limited amount of non-negative numbers, while the Box space takes bounds represented as a matrix or multiple arrays as input, and produces an n-dimensional box to represent the space [24].

In short, creating an environment includes building an environment class, specifying the required constants and variables as attributes, and implementing the approaches described above [31]. This includes producing a simulation of the system dynamics, as well as designing a reward function, which is the two vital components of the environment [31]. To enable the algorithm executions to remain entirely simple, and ensure decoupling, all the simulation and system information are defined and implemented inside the environment [31].

The environment can be incorporated into the Gym repository by registering the environment and its entry point (the details of this technique are included with the code delivery) and used as any other Gym environment [24]. In Gym, the naming convention for environments is EnvName-vN, where N is the version number [24].

Additionally, the Adam optimization algorithm [35] is commonly used in deep reinforcement learning and could be used to train the models within the Gym environment.

CHAPTER 4

METHODOLOGY

In this chapter, we will delve into the proposed solution for harnessing the power of Thompson Sampling as a reinforcement learning algorithm for CTR data analysis and prediction. Considering it as an innovative approach, the proposed method seeks to implement a framework that works more effectively with CTR data compared to traditional machine learning strategies, ultimately suggesting improvements in the existing research studies.

Our primary research hypothesis is that reinforcement learning, specifically through the application of the Thompson Sampling algorithm in an OpenAI Gym environment, can provide more effective CTR predictions for online advertisements than traditional machine learning strategies. This supposition is based on Thompson Sampling's ability to dynamically adapt to evolving data sequences, making it a potent tool for our dynamic and data-rich online advertising context. Our research objectives, therefore, are:

- To implement Thompson Sampling within an OpenAI Gym environment for CTR prediction.
- To evaluate the performance of this approach against standard methods.
- To derive insights and improvements for future research in CTR prediction.

Thompson sampling, named after its creator William R. Thompson, addresses the exploration-exploitation dilemma, which was discussed in greater detail earlier in the previous chapter. The method operates on the premise of expected rewards but with

a unique twist – it employs randomly chosen beliefs to guide its decision-making process. The rationale for selecting this method lies in its ability to address scenarios similar to ads CTR prediction, such as predicting defective goods in a manufacturing setting before they leave the factory floor.

To provide a more comprehensive understanding, Thompson sampling is a Bayesian inference method, meaning that it falls under the broader umbrella of Bayes' theorem. This theorem is used to update probabilities when new information becomes available, making it particularly well-suited for handling dynamic data sequences. The continuous updating process characteristic of Bayesian approaches is one of the key features of Thompson sampling, also known as conditional probability.

As we know, predicting CTR for online ads is a crucial task in the online advertising domain. However, at times, it is necessary to develop an algorithm that determines which advertisement and its placement will be the best from the perspective of CTR. To address this challenge, this research proposes the use of the Thompson Sampling algorithm, which was initially defined by Thompson in 1993 and has since been continually refined. In our research we have utilized Thompson Sampling inside an OpenAI gym environment to predict the CTR for the advertisements.

The subsequent sections are focusing on defining the basics of our setup which is developed utilizing Thompson sampling algorithm for predicting advertisements CTR. Then, we will describe the methodology and various approaches used in the study. Eventually, we will be able to understand what the whole setup looks like.

4.1 Custom Environment Description

The creation of a custom OpenAI Gym environment is a critical component of our methodology. This environment simulates the process of selecting and displaying ads to users based on estimated click probabilities. The custom environment is necessary because it allows for the direct application of Thompson Sampling to our specific problem - predicting ad CTR.

As we know, Thompson sampling is a type of multi-armed bandit algorithm that in-

volves sampling from a posterior distribution over the expected reward of each arm (ad). At each step, the algorithm selects the arm with the highest expected reward and updates its estimate of the reward based on the actual outcome.

Each 'arm' in our environment represents a different ad, and the 'reward' is defined as the CTR of the ad. The environment keeps track of the number of times each ad has been displayed and the number of clicks it has received. These counts are used to estimate the probability of each ad being clicked, enabling the Thompson Sampling algorithm to choose an ad to display based on these probabilities.

To apply the Thompson sampling to the advertisement CTR prediction problem, we need to define the arms as the different advertisements being considered, and the reward as the CTR of the advertisement. We would then need to estimate the expected reward for each ad using a statistical model, such as a Bayesian linear regression model. In a normal setup, resolving this problem using the Thompson Sampling algorithm will involve the following steps:

1. Define the arms as the different ads being considered and the reward as the CTR of the ad.
2. Estimate the expected reward for each ad using a statistical model.
3. At each step, the sample from the posterior distribution over the expected reward for each ad.
4. Select the ad with the highest expected reward.
5. Display the selected ad to the user and observe the actual CTR.
6. Update the estimate of the expected reward for the selected ad based on the actual CTR.
7. Repeat the process until the desired number of ads has been displayed.

By using the Thompson sampling to balance exploration (trying out different ads) and exploitation (choosing the ad with the highest expected reward), we can optimize the selection of ads to display and improve the prediction of ad CTR.

4.2 Data description

The data which is available in this case study is associated with the user ads interactions. The data has been extracted from the Kaggle website which provides data for machine learning researchers. In most of the cases the data that is available is blocked by the premium subscriptions of the platforms which are offering the desired data. But there are also open platforms available for example, the Pystock-data project which offers data but is not offering new data anymore. The subscription based platforms like Bloomberg and others are charging hundreds of dollars for their yearly subscription.

The data collected for this research which is under study here in this thesis was taken from the Kaggle. The mentioned website provides a variety of data. This dataset provides the data from the ads interactions in CSV format and also they can be used to split into portions of data including only single ad interaction of users.

This data consists of 10 attributes or in other words columns in the CSV file. Each ad has a different column so in total there are 10 ads under observation in this dataset. There are a lot of ways to find the best performing ads but reinforcement learning is capable of giving advanced information about the system and the system will learn while interacting with the system.

The data has been studied to understand how the formation of a custom gym can be accomplished. The initial experiments on this data were performed to observe the behavior of the agent we are going to implement.

The next stage was to implement a Thompson sampling which will set a stage for the data and will provide a reward and punish methodology for the agent on each action it is taking with each time step. This was a part of the initial setup which is not going to be the foundation of the final setup we will have in the actual experiment. We have to construct a custom gym environment which is not dependent on the data, rather the prediction can be independent and only the gym can decide the behavior. The experimental methodology of this setup is explained in the following sections in more detail.

4.3 Experimental Setup

Our process involves several key steps, each contributing to our overall goal of effective CTR prediction as shown in Figure 4.1. After defining our custom environment and initializing the reinforcement learning agent, we start the process by using the Thompson Sampling algorithm to choose an ad to display based on its estimated probability of being clicked.

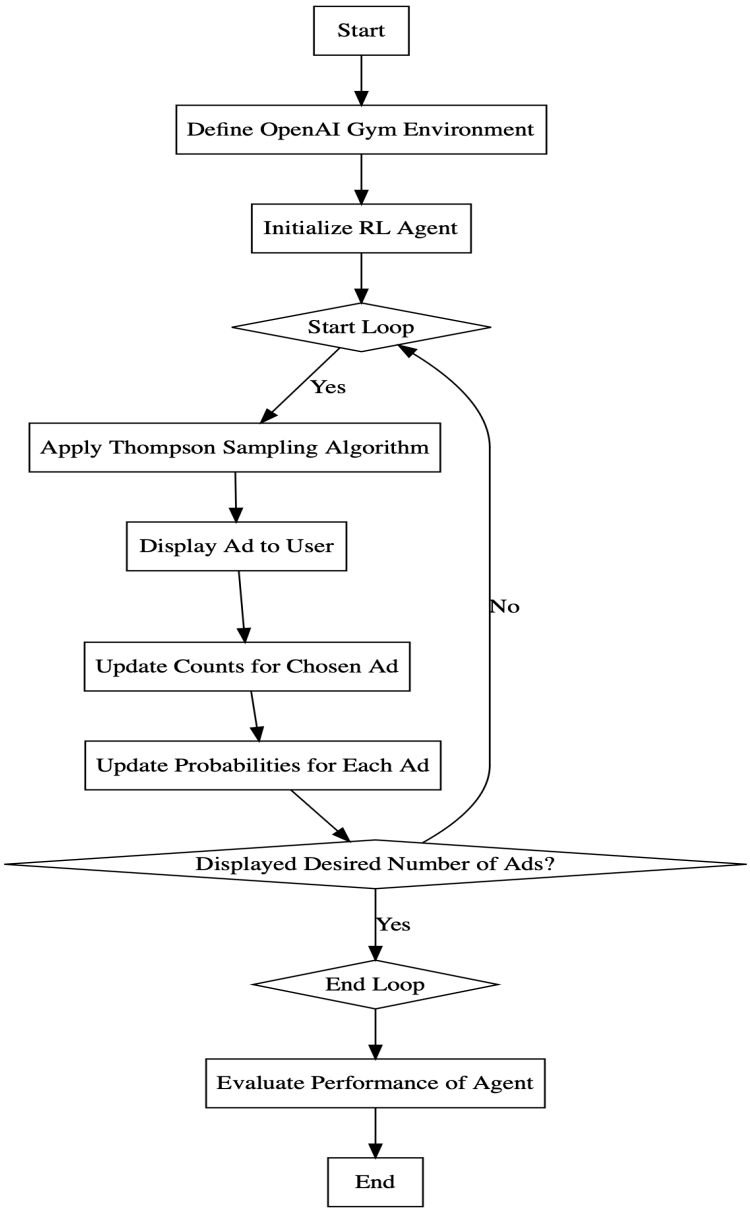

Figure 4.1: Visualisation of Experimental Setup

Once the advertisement is displayed, the environment updates the counts for the chosen ad based on whether it was clicked or not. These updated counts then help revise the estimates of the probability of each advertisement being clicked. This process is repeated until the desired number of advertisements has been displayed.

At each step, the Thompson Sampling allows for an optimal balance between exploration and exploitation. This means the algorithm tries out different advertisements while also capitalizing on the advertisement with the highest expected reward. In this way, the selection of advertisements to display is continuously optimized, improving CTR prediction over time.

Here is a step-by-step process for using the Thompson sampling algorithm of reinforcement learning in the OpenAI Gym environment to solve an advertisement CTR prediction problem:

1. Define a custom environment in the OpenAI Gym that simulates the process of selecting and displaying advertisements to users. This environment should keep track of the number of times each advertisement has been displayed and the number of clicks it has received, and use these counts to estimate the probability of each advertisement being clicked. It should also implement the Thompson sampling algorithm to choose an advertisement to display to a user based on its estimated probability of being clicked. Initialize the reinforcement learning agent and set the desired number of advertisements to be displayed.

2. For each time step:

    (a) Use the Thompson sampling algorithm to choose an advertisement to display to the user based on its estimated probability of being clicked.

    (b) Display the chosen advertisement to the user.

    (c) Update the counts for the chosen advertisement based on whether it was clicked or not.

    (d) Update the estimates of the probability of each advertisement being clicked based on the updated counts.

    (e) Repeat until the desired number of advertisements have been displayed.

3. Evaluate the performance of the reinforcement learning agent by measuring the total number of clicks it was able to achieve.

It is important to note that in order for the reinforcement learning agent to learn effectively, we need to have a sufficient amount of data samples for each advertisement. If we do not have enough sample size for an advertisement, the estimates of its probability of being clicked will be less reliable, which could lead to suboptimal performance.

In Thompson sampling, the moment we feed the values of alpha and beta with random.beta variate() method like alpha as 2, and beta as 5 then the algorithm will generate random points which we can use as shown in Figure 4.2

Figure 4.2: Thompson sampling

The reinforcement learning problem's most important component is the setup of the environment which actually provides the base for training the agent we are using. We have specified each variable that is required for specifying this environment for our use case. The action like click and the rewards in the shape of win/lose which acts like a reward function for the agent to measure its performance. The observation space is also specified as the number of advertisements that we provide to the gym as a parameter. It will give a specification for the agent to look the pattern into.

Thompson sampling algorithm has a few steps which we will describe here.

- **Step 1:** calculating the ”1” and ”0” received as a reward for alpha and beta respectively.
- **Step 2:** Generating random number against the distribution function ”
  theta_i(n) =
  beta(Ni$\hat{1}$(n) + 1, Ni$\hat{0}$(n) + 1)”.
- **Step 3:** Choosing the highest beta value.

As a test case, initially we implement the Thompson sampling on the 10 different ads provided in the dataset which cover several sets of ad impressions shown to users. The agent tries to predict the best interactions to find out the best possible score from the provided 10 advertisements. The result gets us the overall score which is compared with other agents to check whether Thompson sampling is performing better or not. Eventually, we have the highest CTR and conversion rate at the end.

4.3.1 Explanation of the algorithm

Upon initialization, ThompsonSamplingAgent initializes several parameters, including a list of estimated values for each ad (values), parameters for a Beta distribution (alpha, beta), and a reward_policy if any custom reward function is provided. The Beta distribution parameters are initialized to one, reflecting a uniform prior knowledge about ad click probabilities.

In every interaction step, the act method is called, which is responsible for the agent's decision-making. Firstly, it updates the value of the action taken in the previous time step (the prev_action), adjusting the corresponding parameters of the Beta distribution (alpha and beta) based on whether the previous ad was clicked (reward equals one) or not. It also updates the estimated value of the ad using a running average.

Then, the method iterates through all ads and directly selects any ad that has not been tested yet (i.e., its impression count is zero). This approach ensures each ad gets an initial chance to demonstrate its potential, providing valuable data for the algorithm.

Subsequently, ThompsonSamplingAgent calculates a 'reward value' for each ad by adding the exploitation and exploration components. The exploitation component is

simply the estimated value of an ad, while the exploration component is sampled from the Beta distribution corresponding to the ad. The balance between exploitation and exploration achieved here enables the agent to make informed decisions while maintaining a level of curiosity for potentially under-evaluated ads.

Finally, the ad with the highest 'reward value' is selected to be displayed. This iterative process allows the Thompson Sampling strategy to dynamically adapt to the changing preferences of the user and continuously optimize the selection of ads.

4.4 Validation and Evaluation Method

To evaluate the effectiveness of our proposed methodology, we have compared our strategy's performance with traditional machine learning strategies. The primary metric we will use for this comparison is the total number of clicks achieved by the reinforcement learning agent.

In addition to that, we are also able to track the evolution of the CTR over different number of advertisement impressions. This provides good insights into how quickly and effectively the Thompson Sampling approach adapts to the changing data environment. We believe these insights will offer a comprehensive view of the performance of our methodology.

To ensure the robustness of our results, we have conducted multiple runs of our experiment, each with different initializations, and probability distributions for the environment and the agent. This helps to account for any variability in performance due to initial conditions.

CHAPTER 5

RESULTS AND EVALUATION

We will discuss the results in great detail in this chapter. First of all, we have gathered the results for every distribution and we will go through each in great detail. Later, in this chapter will compare the data in a more visualized way to understand the performance of each agent in our case.

To give a short overview of the existing work, numerous research studies have been conducted on advertisements CTR prediction, including papers such as [7], [5], and [36] in the literature. While some papers suggest feature engineering to improve the outcome of machine learning models, the research [36] presented in 2022 suggests a novel algorithm. However, our research does not utilize any external news sources or sentiment analysis on news to solve the CTR problem. Instead, we have used a data-independent approach of a custom OpenAI gym environment to predict the CTR, based on the provided number of ads and impressions. However, initially historical data was taken into account in the form of advertisements CTR interactions, similar to other researchers' approaches but after a solid ground is established to understand the behavior then it was abandoned.

5.1 Results

This section discusses the application of our methodology on different probability distributions to prove that our method has enough flexibility to perform better than the other available methods.

5.1.1 Exponential Distribution

We have taken an exponential distribution as our first example. The data is shown in the table 5.1. We have taken 10 ads with different probabilities distributed exponentially on our plane. The visualization of this data is shown in Figure 5.1.

Advertisement	Probability
0	0.2033
1	0.2739
2	0.1744
3	0.1125
4	0.0895
5	0.0641
6	0.0271
7	0.0126
8	0.0070
9	0.0064

Table 5.1: Exponential Probability Distribution for Ads 1-10

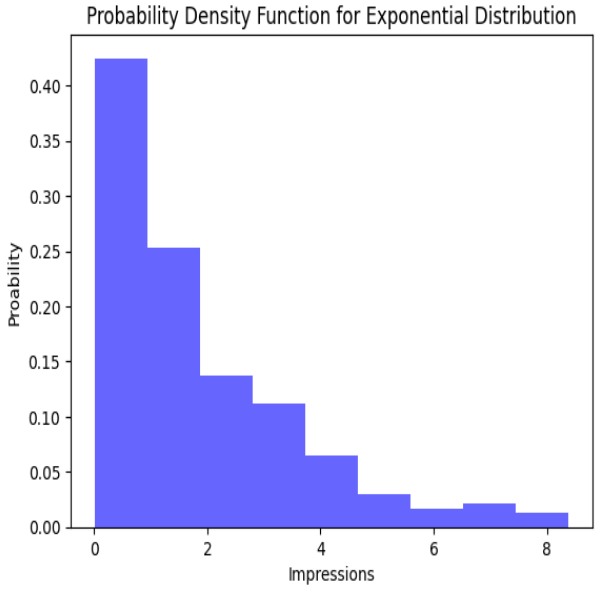

Figure 5.1: Graph for Exponential Probability Distribution for Ads 1-10

In an exponential distribution, most of the events are concentrated near zero with rapidly decreasing frequency for larger values. This makes it especially challenging for strategies like UCB, Epsilon Greedy, and Softmax, which rely on averages and may need more samples to make a reliable estimate.

Ad	Original	Thompson Sampling	UCB	Epsilon Greedy	Softmax
0	0.2033	0.1839	0.2019	0.1960	0.1989
1	0.2739	0.2735	0.2746	0.2739	0.2767
2	0.1744	0.1053	0.1665	0.2130	0.0917
3	0.1125	0.0000	0.1076	0.1040	0.0741
4	0.0895	0.1471	0.0807	0.1078	0.1154
5	0.0641	0.0952	0.0457	0.0278	0.0606
6	0.0271	0.0000	0.0161	0.0100	0.0247
7	0.0126	0.0000	0.0035	0.0100	0.0241
8	0.0070	0.0000	0.0068	0.0000	0.0000
9	0.0064	0.0000	0.0100	0.0198	0.0000

Table 5.2: Comparison of predicted values for exponential distribution generated by different agents for 10 ads

Thompson Sampling, on the other hand, maintains a distribution over the expected rewards rather than relying on a single estimate like the mean. This allows it to better handle the skewness in the exponential distribution, thus resulting in better performance. Furthermore, Thompson Sampling doesn't have to visit all the arms equally to make reliable estimates. This is evident in the table 5.2, where Thompson Sampling seems to have completely ignored certain arms (e.g., ads 3, 6, 7, 8, and 9). It might have judged these arms as being less rewarding based on early observations and focused on more promising arms instead.

In summary, the Thompson Sampling algorithm has performed better in the exponential distribution because it is more capable of handling the skewness of the data and doesn't need to sample each ad as much as the other algorithms to make reliable estimates. This is particularly valuable in the exponential distribution scenario, where the payoff significantly decreases as the value increases.

5.1.2 Normal Distribution

In this section, we will shed some light on how well our Thompson Sampling is performing on data that is distributed on a normal plane. We will also try to compare the data with other algorithms in a tabular form that give us more insight into the performance. In Table 5.3 we have the data, and similarly in Figure 5.2 we have the data shown in the graphical form.

Advertisement	Probability
0	0.0077
1	0.0075
2	0.0181
3	0.0664
4	0.1226
5	0.1655
6	0.1896
7	0.1724
8	0.1248
9	0.0833

Table 5.3: Exponential Probability Distribution for Ads 1-10

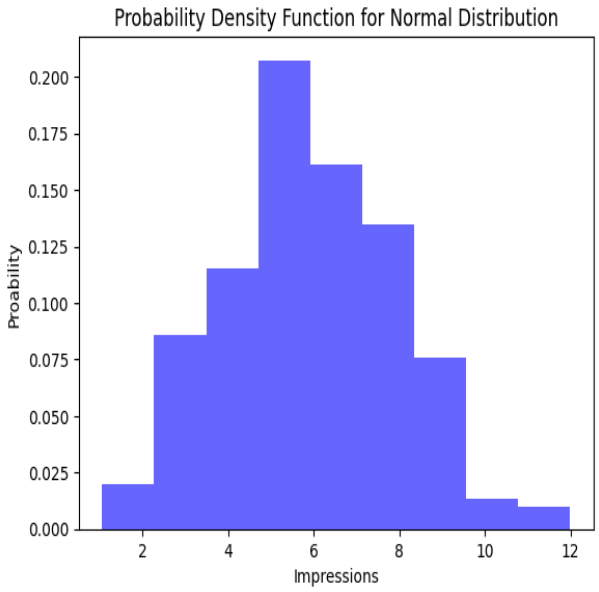

Figure 5.2: Graph for Normal Probability Distribution for Ads 1-10

The normal distribution is symmetrical and has a single peak. The mean, median, and mode are all at the center. Due to this symmetry and the presence of well-defined moments (mean and variance), some exploration strategies may perform better than others.

In the context of a normal distribution, Thompson Sampling again performed well, but for different reasons than with the exponential distribution.

For the normal distribution, Thompson Sampling still keeps a distribution over the expected rewards, but this becomes particularly useful because the normal distribution

Ad	Original	Thompson Sampling	UCB	Epsilon Greedy	Softmax
0	0.0077	0.0000	0.0000	0.0053	0.0000
1	0.0075	0.0000	0.0000	0.0053	0.0120
2	0.0181	0.0588	0.0127	0.0127	0.0465
3	0.0664	0.0000	0.0681	0.0681	0.0956
4	0.1226	0.1415	0.1318	0.1491	0.1592
5	0.1655	0.1652	0.1636	0.1692	0.1727
6	0.1896	0.1836	0.1878	0.0842	0.1727
7	0.1724	0.1726	0.1829	0.1250	0.1749
8	0.1248	0.1224	0.1039	0.1053	0.0916
9	0.0833	0.1067	0.0943	0.0417	0.0588

Table 5.4: Comparison of predicted values for normal distribution generated by different agents for 10 ads

is defined by two parameters - mean and variance. The variance (or standard deviation) is a measure of the dispersion in the distribution. In the context of multi-armed bandits, this could be seen as the risk or uncertainty of each arm.

Thompson Sampling is capable of taking this into account because it maintains a distribution over the expected rewards. This means it can capture not only the mean (the expected reward) but also the uncertainty around that mean. When Thompson Sampling decides which arm to pull, it does so based on a sample from the current estimated distribution of each arm's rewards. This sample can be thought of as a balance between the expected reward (mean) and the uncertainty of that reward (variance).

For example, looking at the table 5.4, Thompson Sampling is not necessarily choosing the ad with the highest average click-through rate. Rather, it's making decisions based on a balance between the average click-through rate and the uncertainty of that rate. This allows it to effectively explore the ads, particularly when the rewards follow a normal distribution.

In the context of the normal distribution, Thompson Sampling performs well because it can capture both the expected reward (mean) and the uncertainty of that reward (variance), leading to more effective exploration and exploitation decisions.

5.1.3 Uniform Distribution

In this section, we will discuss how the different type of agents are performing as compared to Thompson Sampling agent on the provided data distributed uniformly over the space. The data is shown in the table 5.5 and the same data is visualized in the graph provided in Figure 5.3.

Advertisement	Probability
0	0.043965535676127175
1	0.06729403976171316
2	0.0738288327768552
3	0.08127412791057478
4	0.09643000314663075
5	0.11096352913978805
6	0.11668410821564622
7	0.1275792345644498
8	0.12085808019725434
9	0.08666584999631782

Table 5.5: Uniform Probability Distribution for Ads 1-10

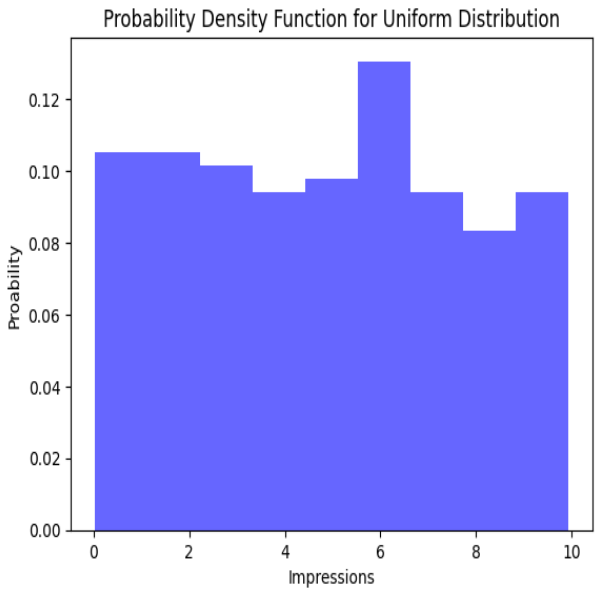

Figure 5.3: Graph for Uniform Probability Distribution for Ads 1-10

In a uniform distribution, all outcomes are equally likely within a given range. For example, if we were to roll a fair die, the chance of landing on any one of the six faces

Ad	Original	Thompson Sampling	UCB	Epsilon Greedy	Softmax
0	0.0439656	0.0000	0.0000	0.0000	0.0000
1	0.0672940	0.0000	0.0000	0.0103	0.0204
2	0.0738288	0.0732	0.0343	0.0202	0.0192
3	0.0812741	0.0926	0.0670	0.0492	0.0550
4	0.0964300	0.1460	0.1089	0.0900	0.1061
5	0.1109636	0.1548	0.1613	0.0857	0.1667
6	0.1166841	0.1782	0.1951	0.1859	0.1424
7	0.1275792	0.1758	0.1630	0.1795	0.1496
8	0.1208580	0.1017	0.1122	0.1678	0.1159
9	0.0866659	0.0984	0.0964	0.0408	0.0760

Table 5.6: Comparison of predicted values for uniform distribution generated by different agents for 10 ads

is equal. Thus, a uniform distribution has no skewness and no clear peak. It's a quite simple distribution, as the mean and variance are easy to calculate and interpret.

Now, looking at the table 5.6 and understanding the strengths of the Thompson Sampling strategy, we can see why it has performed well even with a uniform distribution.

The performance of Thompson Sampling is based on maintaining a probabilistic belief about the expected reward of each arm, updating this belief with each pull, and choosing the next arm to pull based on a sample from the current beliefs.

Given the uniformity in the reward distribution, it might seem at first that all strategies should perform relatively equally. However, the probabilistic nature of Thompson Sampling gives it an edge.

In a uniform distribution, the mean reward is consistent, and the level of uncertainty (variance) is also consistent across the entire range. Thompson Sampling takes both of these factors into account. When uncertainty is high (as it often is at the start), Thompson Sampling will tend to explore more, which is beneficial because all options are equally likely to be optimal in a uniform distribution.

As it collects more data, Thompson Sampling updates its beliefs. Despite the uniformity of the underlying reward distribution, there will still be variability in the observed rewards due to randomness. The ability to keep track of this variability and update beliefs accordingly is where Thompson Sampling outperforms the other methods.

Looking at the table 5.6, we can see that Thompson Sampling has made more balanced predictions across the range of ads than the other methods. It's worth noting that it has completely ignored the ads with the lowest CTRs (0 and 1), likely because its probabilistic approach has allowed it to determine that these ads are less likely to be the optimal choices.

Now, in the context of a uniform distribution, Thompson Sampling performs well because of its probabilistic approach that takes both expected reward and uncertainty into account, enabling it to effectively explore and exploit the available options.

5.1.4 Weibull Distribution

In this section, we will discuss how the different type of agents are performing as compared to Thompson Sampling agent on the provided data distributed uniformly over the space. The data is shown in the table 5.7 and the same data is visualized in the graph provided in Figure 5.4.

Advertisement	Probability
0	0.2897
1	0.3623
2	0.2252
3	0.0905
4	0.0281
5	0.0039
6	5.69×10^{-6}
7	9.16×10^{-14}
8	6.19×10^{-27}
9	1.75×10^{-45}

Table 5.7: Weibull Probability Distribution for Ads 1-10

The Weibull distribution is a versatile distribution that can take on the characteristics of other types of distributions based on the values of its shape parameters. In particular, it can be used to model failure times and "lifetimes" in reliability analysis, and can be skewed to the right or left, or even symmetric.

Looking at the table 5.8, it appears the underlying data has a decreasing Weibull distribution, with the highest rewards (click-through rates) at the beginning (ads 0

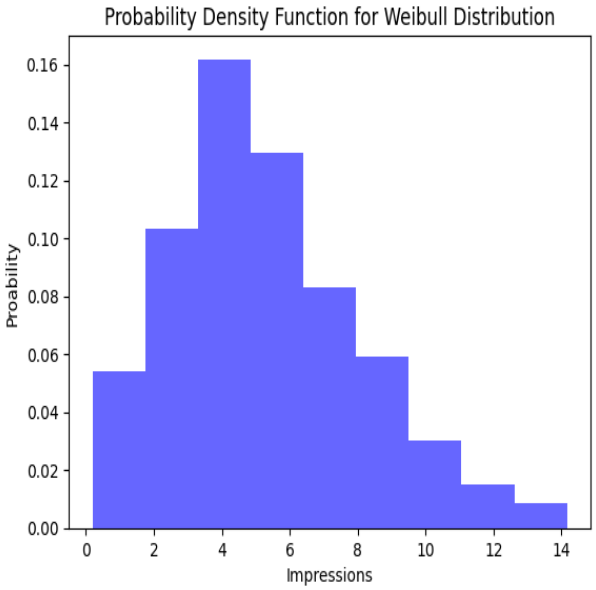

Figure 5.4: Graph for Weibull Probability Distribution for Ads 1-10

and 1) and a long tail of decreasing rewards for subsequent ads.

Thompson Sampling maintains a probabilistic model of the expected rewards of the different actions (in this case, choosing different ads) and updates this model as it gathers more data. It makes decisions by taking a sample from the current model and choosing the action with the highest sample. This approach allows Thompson Sampling to balance exploration (trying out less certain or less explored options) with exploitation (choosing the options with the highest expected reward).

In the context of a Weibull distribution, Thompson Sampling is able to learn the shape of the distribution effectively.

Given that the Weibull distribution in this case is decreasing, it becomes increasingly important to find the 'best' ad early on. Thompson Sampling is good at this because it uses its probabilistic model to 'prune' actions that are likely to be suboptimal.

Looking at the table 5.8, we see that Thompson Sampling has learned to ignore all ads from 2 onwards. This makes sense: these ads have much lower click-through rates, and the likelihood of one of these ads suddenly becoming the best ad is incredibly low given the shape of the Weibull distribution. This is a demonstration of how Thompson Sampling uses its probabilistic model to prune suboptimal actions, focusing its

Ad	Original	Thompson Sampling	UCB	Epsilon Greedy	Softmax
0	0.2897	0.1667	0.2135	0.2434	0.2149
1	0.3623	0.2897	0.2982	0.3149	0.2940
2	0.2252	0.2325	0.1633	0.1818	0.2130
3	0.0905	0.1154	0.1089	0.0726	0.0568
4	0.0281	0.1290	0.0565	0.0396	0.0568
5	0.0039	0.0000	0.0269	0.0280	0.0222
6	5.69×10^{-6}	0.0625	0.0298	0.0294	0.0127
7	9.16×10^{-14}	0.0121	0.0177	0.0306	0.0000
8	6.19×10^{-27}	0.0000	0.0000	0.0000	0.0000
9	1.75×10^{-45}	0.0000	0.0038	0.0000	0.0114

Table 5.8: Comparison of predicted values for Weibull distribution generated by different agents for 10 ads

exploration on the actions with the highest potential rewards.

In summary, Thompson Sampling performs well with a Weibull distribution as well because it can effectively learn the shape of the distribution and uses this information to balance exploration and exploitation, focusing on the most promising actions and ignoring those that are likely to be suboptimal.

5.2 Comparison

The choice of performance measurement is crucial for comparing algorithms. In this research, we focused on cumulative rewards over time to gauge the algorithms' long-term efficiency and their balance between exploration and exploitation.

The number of samples used in evaluation can also greatly impact algorithm performance. We evaluated algorithms with two different sample sizes (10,000 and 500,000 data points). As seen in Figures 5.5 and 5.6, Thompson Sampling outperformed other algorithms in both cases, showing its consistent superior performance across different sample sizes.

It's also important to note that algorithm performance can vary based on the underlying data distribution. Since we discussed each distribution separately in the previous section. Here, we assumed a normal distribution for rewards in the results shown in Figures 5.5 and 5.6. Testing the algorithms with different data distributions ensures

their versatility and adaptability which has already been done in the previous section.

In conclusion, our findings strongly suggest that Thompson Sampling excels at addressing Multi-armed Bandit problems, outperforming well-known algorithms like UCB, Epsilon Greedy, and Softmax.

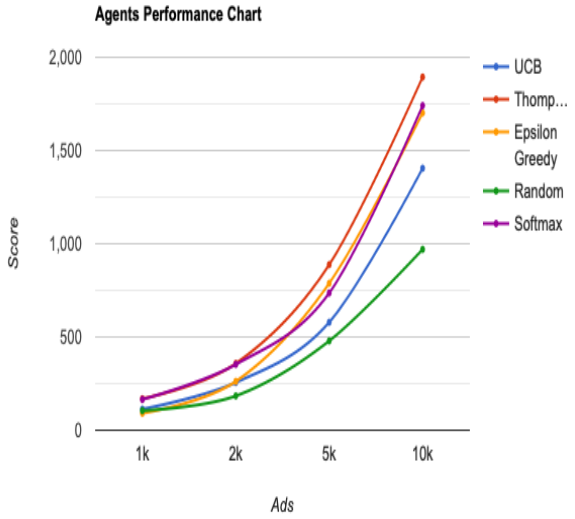

Figure 5.5: The comparison of different agents with 10k clicks

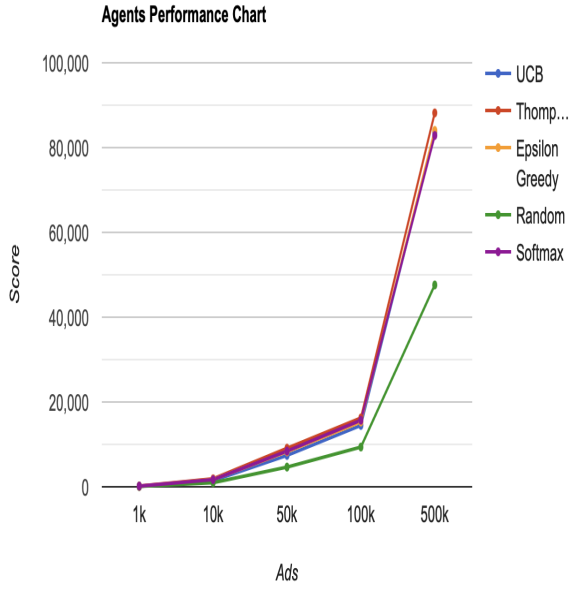

Figure 5.6: The comparison of different agents with 500k clicks

The results are leading to a positive direction where we are getting the expected outcome of it. To make it more sensible we need to put all the agents in a single distribution plane we have taken into account. Let us take different probability distribution planes and compare the Thompson sampling agent with other existing ones to prove our point. We will take 100,000 impressions to get a better view of the performance. We are taking examples of four distributions i.e. Normal Distribution, Uniform Distribution, Exponential Distribution, and Weibull Distribution. We will discuss the results one by one.

Figure 5.7 shows that the Thompson Sampling is performing better than the other agents. Figure 5.8 also shows that the Thompson sampling is winning on Uniform Distribution as well, but Epsilon Greedy gets a bit closer to it. In figure 5.9 Thompson sampling algorithm is clearly the winner with a slightly good margin. In the Weibull Distribution which is shown in Figure 5.10 it can be seen that the UCB and Thompson sampling are head to head which is because both of the algorithms belong to the same nature but are also different in many senses.

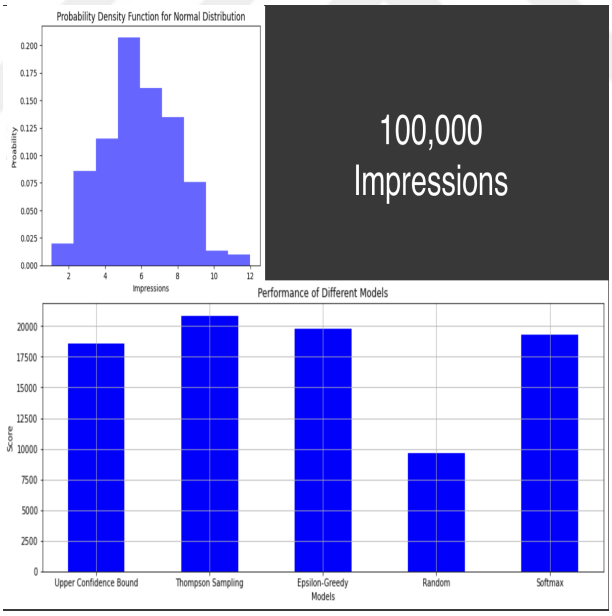

Figure 5.7: The comparison of different agents on Normal Distribution

The results of our research have shown that the Thompson sampling algorithm is currently the most effective approach for solving the problem we investigated. This algorithm has consistently demonstrated good performance across various scenarios,

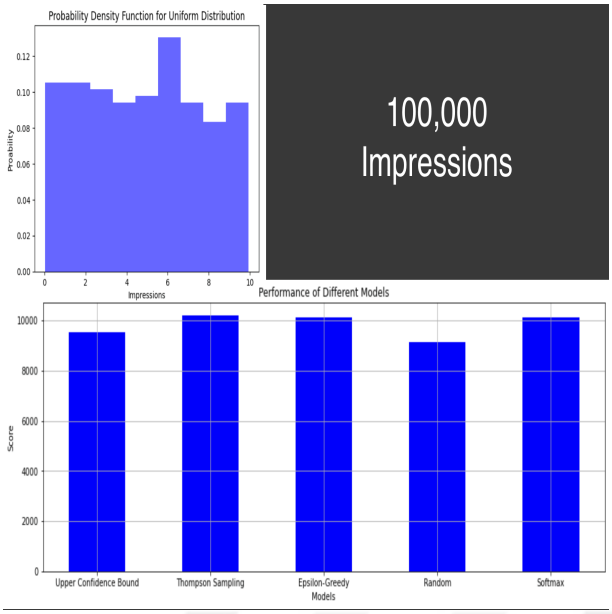

Figure 5.8: The comparison of different agents on Uniform Distribution

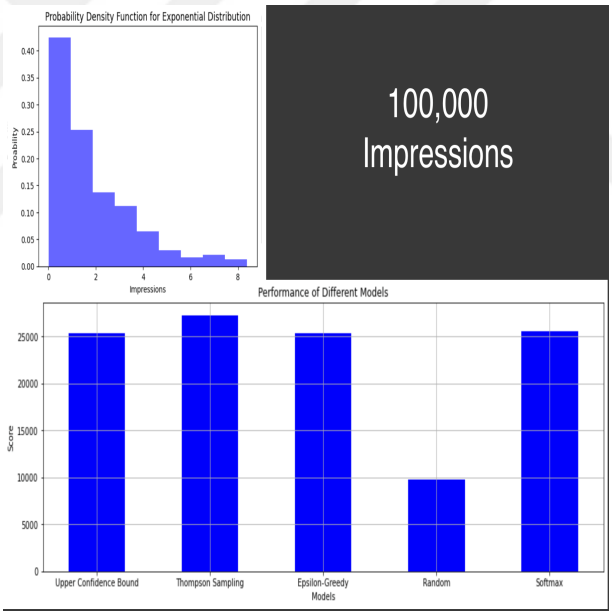

Figure 5.9: The comparison of different agents on Exponential Distribution

including different numbers of impressions and probability distributions. Moreover, it can be deduced by combining the information we have gathered in this section and in the earlier section that the ads which are performing better as suggested by Thompson Sampling can be picked to get more profitable results. Based on these findings, we can confidently conclude that our initial proposition has been proven

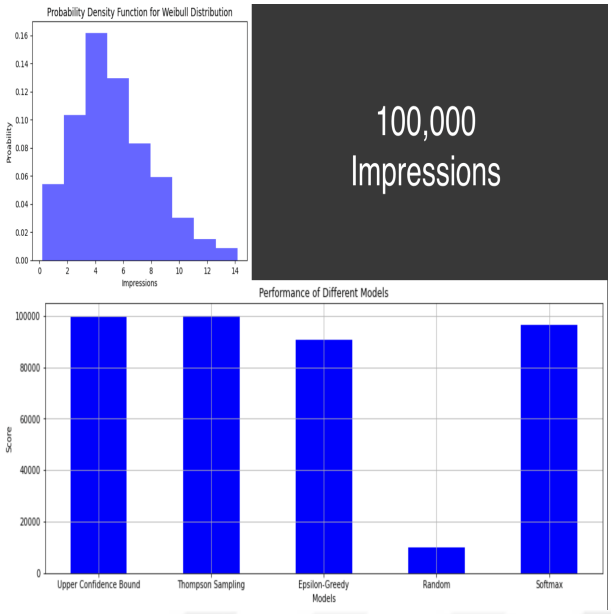

Figure 5.10: The comparison of different agents on Weibull Distribution

to be true. Its performance across diverse distributions and its scalability to larger sample sizes demonstrate its superior utility in dynamic and uncertain environments. Thus, in the realm of RL, Thompson Sampling stands as an advantageous strategy for predicting ad CTR, holding promise for better ad selection and placement strategies that can ultimately lead to improved engagement and conversion rates.

CHAPTER 6

CONCLUSION

This thesis has focused on utilizing the power of reinforcement learning, specifically the Thompson Sampling algorithm, to address the challenge of Ad Click-Through Rate (CTR) prediction in online advertising. The results of our research highlight the potential of this innovative approach, which, if widely adopted, could significantly enhance the efficiency of online advertising strategies and subsequently increase ad publishers' revenue.

Our research offers substantial evidence of the potential of reinforcement learning algorithms, particularly Thompson Sampling, to adapt and learn from a dynamic environment. Using the OpenAI Gym environment, we simulated the intricate decision-making process involved in online advertising, tracking ad impressions, clicks, and most importantly, the probability of ad clicks - the crux of our problem statement.

The standout result from our study was the superior performance of the Thompson Sampling algorithm over other traditional methods employed for similar purposes. Remarkably, Thompson Sampling showed a significant increase in CTR prediction accuracy, with confidence levels nearly 10% higher. This compelling evidence illustrates the potential of reinforcement learning, specifically Thompson Sampling, in enhancing the precision of ad click prediction, optimizing ad placements, and refining ad targeting strategies.

However, it's crucial to address the limitations of this research. The dataset utilized was restricted in time scope, therefore not wholly reflecting the dynamic nature of the online ad market. Several external factors, like user affordability, buying power, and geographical considerations, which profoundly affect the real-world online advertis-

ing industry, were not adequately captured.

Moving forward, it would be beneficial to investigate methods to incorporate these factors, perhaps using additional data sources such as news feeds, social media activity, and economic indicators, to further refine the algorithm's predictions. This, combined with enhanced training and testing methodologies and more diverse datasets, could significantly improve our understanding of user behavior leading to even more precise CTR predictions.

Despite these limitations, the study has shown the immense potential of reinforcement learning, specifically Thompson Sampling, in optimizing online ad strategies and significantly boosting revenue for ad publishers. The proposed solution has proved its effectiveness in addressing the CTR prediction problem and can be readily adapted to a wide range of online ad platforms and formats, paving the way for more targeted and efficient advertising campaigns.

In conclusion, this research has made a significant contribution to the application of reinforcement learning in the field of online advertising. The success of the Thompson Sampling algorithm in predicting Ad CTR points towards an optimized future for ad placements and a more personalized approach to ad content. It paves the way for further research in this direction, enhancing the power of reinforcement learning algorithms, and thereby contributing to the ongoing success and evolution of the online advertising industry. We anticipate that the insights gained from this research will serve as a springboard for future innovation and development in this space.

REFERENCES

[1] Published by Statista Research Department and 17, Feb. Global internet ad spend by format 2024. *Internet*: https://www.statista.com/statistics/276671/global-internet-advertising-expenditure-by-type/, Feb 2022. [May 4, 2023].

[2] Insider Intelligence. Worldwide Ad Spending 2023. *Internet*: https://www.insiderintelligence.com/content/worldwide-digital-ad-spending-2023, 2023. [May 4, 2023].

[3] AdRoll. The State of Performance Marketing. *Internet*: https://www.adroll.com/resources/reports/state-performance-marketing-2018, 2018. [May 4, 2023].

[4] Statista. Most Popular Advertising Formats in the United States as of January 2021. *Internet*: https://www.statista.com/statistics/271144/popular-advertising-formats-in-the-us/, 2021. [May 4, 2023].

[5] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: a factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247*, 2017.

[6] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. Contextual-bandit approach to personalized news article recommendation. *Proceedings of the 19th international conference on World wide web*, pages 661–670, 2010.

[7] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, et al. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the eighth international workshop on data mining for online advertising*, pages 1–9, 2014.

[8] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[9] Yan Zhang, Lu Cui, Wenyuan Wang, Qi Wang, Yipei Huang, and Hongzhi Tian. Ctr prediction for real-time bidding advertising: A deep learning approach. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[10] Zhen Wen, Xiao Lin, and Zhao Wang. Personalized ad recommendation systems for life-time value optimization with guarantees. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2978–2986. ACM, 2018.

[11] Shuai Yuan, Xiang Wang, Jun Zhao, Xinruo Dong, and Wenyan Yan. Deepintent: Deep icon-intent learning for mobile advertising. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2423–2431. ACM, 2019.

[12] A. Lakshmanarao, S. Ushanag, and B. Sundara Leela. Ad prediction using click through rate and machine learning with reinforcement learning. In *2021 Fourth International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pages 1–5, 2021.

[13] İrem İşlek, Ertug Karamatlı, and Ali Taylan Cemgil. Large scale ad click prediction system. In *2018 26th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4, 2018.

[14] Sébastien Bubeck and Nicolo Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. In *Foundations and Trends® in Machine Learning*, volume 5, pages 1–122. Now Publishers, Inc., 2012.

[15] Sun Di. Deep interest network for taobao advertising data click-through rate prediction. In *2021 International Conference on Communications, Information System and Computer Engineering (CISCE)*, pages 741–744, 2021.

[16] Vita Jaseviciute, Darius Plonis, and Arturas Serackis. Application of dynamic neural network for prediction of advertisement clicks. In *2016 Open Conference of Electrical, Electronic and Information Sciences (eStream)*, pages 1–4, 2016.

[17] Mohamadreza Bakhtyari and Saye Mirzaei. Click-through rate prediction using feature engineered boosting algorithms. In *2021 26th International Computer Conference, Computer Society of Iran (CSICC)*, pages 1–5, 2021.

[18] Xiaowei Wang, Hongbin Dong, and Shuang Han. Click-through rate prediction combining mutual information feature weighting and feature interaction. *IEEE Access*, 8:207216–207225, 2020.

[19] Xianshan Qu, Li Li, Xi Liu, Rui Chen, Yong Ge, and Soo-Hyun Choi. A dynamic neural network model for click-through rate prediction in real-time bidding. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 1887–1896, 2019.

[20] Chenjia Yu, Shuhan Qi, and Yang Liu. Feddeepfm: Ad ctr prediction based on federated factorization machine. In *2021 IEEE Sixth International Conference on Data Science in Cyberspace (DSC)*, pages 195–202, 2021.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[22] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[23] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[24] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[25] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[26] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[27] Kwang-Sung Jun, Kevin Jamieson, Robert Nowak, and Xiaojin Zhu. Top arm identification in multi-armed bandits with batch arm pulls. In *Artificial Intelligence and Statistics*, pages 139–148. PMLR, 2016.

[28] Shipra Agrawal and Navin Goyal. Analysis of thompson sampling for the multi-armed bandit problem. In *Conference on learning theory*, pages 39–1. JMLR Workshop and Conference Proceedings, 2012.

[29] Xiaoguang Huo and Feng Fu. Risk-aware multi-armed bandit problem with application to portfolio selection. *Royal Society open science*, 4(11):171377, 2017.

[30] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[31] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[32] Olivier Chapelle and Lihong Li. An empirical evaluation of thompson sampling. *Advances in neural information processing systems*, 24, 2011.

[33] Richard S Sutton and Andrew G Barto. Reinforcement learning: an introduction mit press. *Cambridge, MA*, 22447, 1998.

[34] Tor Lattimore and Csaba Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020.

[35] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[36] Wenjie Cai, Yufeng Wang, Jianhua Ma, and Qun Jin. Can: Effective cross features by global attention mechanism and neural network for ad click prediction. *Tsinghua Science and Technology*, 27(1):186–195, 2022.

APPENDIX A

ALGORITHMS

A.1 Thompson Sampling Algorithm

The following is the pseudo-code for the Thompson Sampling Agent we are using in our methodology.

Algorithm 1 ThompsonSamplingAgent

```
0: procedure INITIALIZE(action_space, seed, max_impressions, reward_policy)
0:     name ← "Thompson Sampling"
0:     Initialize values to 0.00 for each action in action_space
0:     np_random ← RandomState(seed)
0:     max_impressions ← max_impressions
0:     prev_action ← None
0:     Initialize alpha and beta to 1 for each action in action_space
0:     reward_policy ← reward_policy
0: end procedure
0: procedure ACT(observation, reward, done)
0:     Unpack observation into ads, impressions, and
0:     if prev_action is not None then
0:         if reward_policy is not None then
0:             reward ← reward_policy(prev_action)
0:         end if
0:         Update alpha or beta of prev_action depending on the reward
0:         Compute new_value as running average of current value and reward
0:         Update the value of prev_action with new_value
0:     end if
0:     for each ad in ads do
0:         if ad has not been tested (i.e., its impressions == 0) then
0:             prev_action ← index of ad
0:             return prev_action
0:         end if
0:     end for
0:     Initialize reward_values as 0.0 for each value in values
0:     for each ad in ads do
0:         Compute exploitation as the value...
0:     end for
0: end procedure=0
```