UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

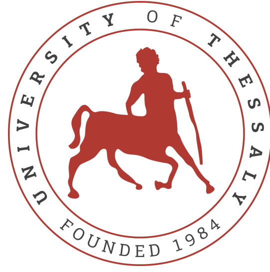
Recommender Systems using Deep Reinforcement Learning

Diploma Thesis

Vasileios Stergiou

Supervisor: Michael Vassilakopoulos

June 2022



UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Recommender Systems using Deep Reinforcement Learning

Diploma Thesis

Vasileios Stergiou

Supervisor: Michael Vassilakopoulos

June 2022

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Συστήματα Συστάσεων με χρήση Βαθιάς Ενισχυτικής Μάθησης

Διπλωματική Εργασία

Βασίλειος Στεργίου

Επιβλέπων: Μιχαήλ Βασιλακόπουλος

Ιούνιος 2022

Approved by the Examination Committee:

Supervisor **Michael Vassilakopoulos**
Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Dimitrios Rafailidis**
Associate Professor, Department of Electrical and Computer Engineering, University of Thessaly

Member **Eleni Tousidou**
Laboratory Teaching Staff, Department of Electrical and Computer Engineering, University of Thessaly

Acknowledgements

First and foremost, I would like to thank my thesis supervisor, Prof. Michael Vassilakopoulos, for his invaluable support and guidance during the development of this thesis and for his input in the selection of the topic.

Finally, I would like to express my gratitude to my family, friends and loved ones who endured this process with me, always offering support and love.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

«Being fully aware of the implications of copyright laws, I expressly state that this diploma thesis, as well as the electronic files and source codes developed or modified in the course of this thesis, are solely the product of my personal work and do not infringe any rights of intellectual property, personality and personal data of third parties, do not contain work / contributions of third parties for which the permission of the authors / beneficiaries is required and are not a product of partial or complete plagiarism, while the sources used are limited to the bibliographic references only and meet the rules of scientific citing. The points where I have used ideas, text, files and / or sources of other authors are clearly mentioned in the text with the appropriate citation and the relevant complete reference is included in the bibliographic references section. I also declare that the results of the work have not been used to obtain another degree. I fully, individually and personally undertake all legal and administrative consequences that may arise in the event that it is proven, in the course of time, that this thesis or part of it does not belong to me because it is a product of plagiarism».

The declarant

Vasileios Stergiou

Diploma Thesis

Recommender Systems using Deep Reinforcement Learning

Vasileios Stergiou

Abstract

Recommender systems have become increasingly popular recently because they can address the problem of information overload by suggesting items of interest to the users. A major drawback of traditional recommender systems is that most of them ignore the dynamic and sequential nature of the recommendation problem. In this thesis we developed a distributed recommender system using Deep Reinforcement Learning based on the Asynchronous Advantage Actor-Critic algorithm (A3C). First, we propose a method for modeling the recommendation problem as an Markov Decision Process using Reinforcement Learning techniques. Then, we present our implementation of A3C algorithm and describe the distributed training procedure. Our system consists of two models: a set of local agents that are trained in parallel by interacting with a local copy of the environment, and a global model whose parameters each local agent updates at the end of a training episode. We have evaluated the algorithm on a known dataset, using popular recommender systems metrics. We compare our experimental results with other related work and show that in many cases it can achieve comparable performance. Finally, we analyze the effect that the number of workers have on training time.

Διπλωματική Εργασία

Συστήματα Συστάσεων με χρήση Βαθιάς Ενισχυτικής Μάθησης

Βασίλειος Στεργίου

Περίληψη

Τα Συστήματα Συστάσεων γίνονται όλο και πιο δημοφιλή επειδή μπορούν να αντιμετωπίσουν το πρόβλημα της υπερφόρτωσης πληροφοριών προτείνοντας στους χρήστες αντικείμενα που τους ενδιαφέρουν. Ένα σημαντικό μειονέκτημα των παραδοσιακών Συστημάτων Συστάσεων είναι ότι τα περισσότερα από αυτά αγνοούν τη δυναμική και διαδοχική φύση του προβλήματος σύστασης αντικειμένων. Η παρούσα Διπλωματική Εργασία προτείνει ένα κατανεμημένο σύστημα συστάσεων με χρήση Βαθιάς Ενισχυτικής Μάθησης με βάση τον αλγόριθμο Asynchronous Advantage Actor-Critic (A3C). Πρώτα, προτείνουμε μια μέθοδο για τη μοντελοποίηση του προβλήματος σύστασης ως Μαρκοβιανή διαδικασία αποφάσεων χρησιμοποιώντας τεχνικές Ενισχυτικής Μάθησης. Στη συνέχεια, παρουσιάζουμε την υλοποίηση του αλγορίθμου A3C και περιγράφουμε την κατανεμημένη διαδικασία εκπαίδευσης. Το σύστημά μας αποτελείται από δύο μοντέλα: ένα σύνολο τοπικών πρακτόρων που εκπαιδεύονται παράλληλα αλληλεπιδρώντας με ένα τοπικό αντίγραφο του περιβάλλοντος και ένα κεντρικό μοντέλο που κάθε τοπικός πράκτορας ενημερώνει τις παραμέτρους του στο τέλος ενός επεισοδίου εκπαίδευσης. Αξιολογήσαμε τον αλγόριθμο σε ένα γνωστό σύνολο δεδομένων χρησιμοποιώντας κοινές μετρικές συστάσεων. Συγκρίνουμε τα πειραματικά μας αποτελέσματα με άλλες σχετικές εργασίες και δείχνουμε ότι, σε πολλές περιπτώσεις, μπορεί να πετύχει συγκρίσιμη απόδοση. Τέλος, αναλύουμε τον αντίκτυπο που ο αριθμός των παράλληλων πρακτόρων έχει στο χρόνο εκπαίδευσης.

Table of contents

Acknowledgements **ix**

Abstract **xii**

Περίληψη **xiii**

Table of contents **xv**

List of figures **xvii**

List of tables **xix**

Abbreviations **xxi**

1 Introduction **1**

1.1 Objective 2

1.2 Methodology 2

1.3 Thesis Structure 3

2 Background **5**

2.1 Recommender Systems 5

2.1.1 Recommender Systems Categorization 7

2.1.2 Evaluation 9

2.2 Deep Learning 11

2.3 Reinforcement Learning 13

2.4 Deep Reinforcement Learning 15

2.4.1 Deep Q Network 16

2.4.2 Policy Gradients 16

2.4.3 Actor Critic 17
2.5 Current Work 18
2.5.1 RL-based Recommender Systems 18
2.5.2 Simulation Environments 19

3 Simulation Environment 21
3.1 Dataset 21
3.2 Formulation of recommendation in RL 22
3.3 Simulation Environment 24
3.3.1 RecSim simulation 25
3.3.2 How ratings are simulated 26

4 Methods 29
4.1 Asynchronous Training using Distributed Agents 29
4.2 Actor-Critic and Advantage 30
4.3 Evaluation Agents 33
4.3.1 Implementation details 34

5 Results 37
5.1 Evaluation Methods 37
5.2 Training Results 38
5.3 Evaluation Results 42
5.3.1 Relevance score 43
5.4 Number of workers and training time 44

6 Conclusions 47

Bibliography 49

List of figures

1.1 Development method of the thesis 2

2.1 Real systems and the recommendations they make 6
2.2 Example of a NxM Utility Matrix in a movie recommender system 7
2.3 Illustration of User-Based CF method 8
2.4 Illustration of Item-Based CF method 8
2.5 Categories of different RS methods 9
2.6 Perceptron 12
2.7 Feed Forward Network 13
2.8 Agent-environment interaction loop 14
2.9 Actor-Critic architecture 17

3.1 Data Flow through components of RecSim [1] 26
3.2 User- movie interaction matrix from MovieLens dataset 27
3.3 User- movie interaction matrix from MovieLens dataset 28

4.1 A3C high level architecture 30
4.2 Advantage Actor-Critic process 31
4.3 The design of Actor-Critic neural network 32
4.4 Advantage Actor-Critic algorithm from [2] 33
4.5 Repository file structure 35
4.6 Class diagram 36

5.1 Random Agent 38
5.2 Actor-Critic Agent 39
5.3 Comparison of A2C and A3C agents 40
5.4 Comparison of A2C and A3C agents 41

5.5 Training statistics illustrating the average CTR per episode for the various models compared in DARES [3] 43
5.6 A3C: Number of workers vs training steps per second. 45
5.7 A2C: Number of workers vs training steps per second. 45

List of tables

3.1 Choices of occupations in the dataset 23
3.2 Features of the MoviesLens dataset used 24

5.1 Comparison of our implemented agents with the models presented in [3] . . 42
5.2 Comparison of our implemented agents with the models presented in [3] . . 43
5.3 Comparison of evaluation metrics for A2C agent against random 44

Abbreviations

A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
API	Application Programming Interface
ANN	Artificial Neural Network
CF	Collaborative Filtering
CPU	Computational Processing Unit
CTR	Click Through Rate
DL	Deep Learning
DNN	Deep Neural Network
DQN	Deep Q-Networks
DRL	Deep Reinforcement Learning
GPU	Graphics Processing Unit
MDP	Markov Decision Process
ML	Machine Learning
NDCG	Normalized Discounted Cumulative Gain
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RS	Recommender System
TD	Temporal-Difference

Chapter 1

Introduction

In the recent past, the increasing importance of the Web as a medium has drastically changed the way we live and experience our lives. A major catalyst in this regard is the rapid expansion and diversity of information available on the Internet and the quick launch of e-commerce. It is becoming increasingly easy to obtain anything (goods, books, movies, news, etc.) over the Internet. A wide range of options may be very tempting for users, but it has been proven that this makes them less motivated to buy a product later [4]. Recommender Systems(RS) serve as a solution to this problem by assisting users find items that interest them using their preferences or reviews of other items. Today, many of the largest companies, such as Google, Amazon, Facebook, and Netflix, use RS in a broad-spectrum of applications, including ads [5], product and news recommendation, and healthcare [6].

Numerous techniques have been preferred to solve the recommendation task, including matrix factorization, collaborative filters, and content-based methods. Despite their success, they suffer from some limitations. They model recommendation as a static process. In this way, they fail to capture the dynamic and sequential nature of recommendation. In a realistic scenario, it is very likely that users' preferences will change over time or that current decisions will influence their future actions. Therefore, studying the recommendation task as a sequential dynamic procedure would be more realistic.

More recently, researchers have explored the potential of framing recommendations as a Markov Decision Process (MDP) by leveraging RL agents to make recommendations. In a sense, both RL and RS share some standard features. They aim to optimize some reward concepts by interacting with an environment and performing actions that lead to updates of internal states.

1.1 Objective

The goal of this thesis is twofold. First, it provides an approach to model the recommendation problem as an MDP by implementing an RL simulation environment. This environment is designed to capture the dynamic interaction between user and item and adapt to user preferences by encoding the dynamics of addiction and boredom in the rewards. Next, we apply Deep Reinforcement Learning by implementing the Advantage Actor-Critic agents to make recommendations. We evaluate the performance of the algorithms using both RL and RS metrics such as average rewards and CTR, NDCG, respectively.

1.2 Methodology

A brief preview of the architecture of the recommendation system is presented in this section. For any reinforcement learning-based technique, we must create an environment based on the formulation of the problem. Because this environment is problem-specific, each solution has its own custom environment tailored to the problem area. During the writing of this thesis, the development method in Figure 1.1, inspired by [7] was followed.

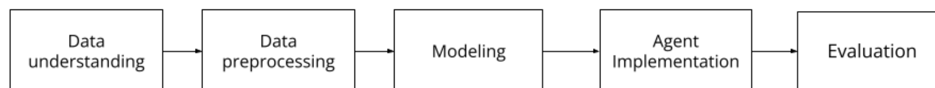


Figure 1.1: Development method of the thesis

- **Data understanding:** to download and understand the structure of the dataset, in our case MovieLens1M.
- **Data preprocessing:** to prepare data for feature extraction.
- **Modeling:** to design Reinforcement Learning simulation environment.
- **Agent implementation:** to implement the A3C agent.
- **Evaluation:** to run various experiments for the agents, and measure the performance.

1.3 Thesis Structure

The following chapters of this thesis are structured as:

Chapter 2 provides some background information in the fields of recommender systems, reinforcement learning, and neural networks critical to the scope of this thesis.

Chapter 3 describes the design and modifications we made to a Reinforcement Learning simulation environment suitable for recommendation problems. We also analyze other existing environments and the challenges they present.

Chapter 4 introduces the implemented agents to mimic the distributed training scenario. A software architecture overview is provided along with some implementation details.

Chapter 5 analyzes the results obtained from the various experiment held.

Finally, Chapter 6 concludes by outlining the findings drawn from our research and providing some directions for future work.

Chapter 2

Background

This chapter is intended to present the background material needed to understand the role of Deep Reinforcement Learning in recommender systems. The models and mathematical equations presented are mainly derived from the work in [8], [9], and [10]. The final section reviews some of the most recent work connected to this thesis.

2.1 Recommender Systems

In life it often happens that we are confronted with circumstances in which we have to make a decision without knowing the options beforehand. In such cases, it seems essential to rely on the advice of experts.

In an effort to emulate this idea, the first paper on Recommender Systems [11] was published and the term Collaborative Filtering was used. Their technique for computing recommendations for a given user was to select items that were preferred by other users similar to the target user. Later, this topic was intensively researched, which led to the extension of the term to recommendation systems to take into account two facts. The system may collaborate with items, not users. The system may suggest articles that might be of interest to users, not just filter them.

As defined in [12], "a recommender system or a recommendation system is a subclass of information filtering systems that attempt to predict the 'rating' or 'preference' a user would give to an item, by optimizing a set of goals." They do so by estimating a metric for interaction between users and items. There are two primary ways to formulate metrics: by calculating the score for a user-item interaction or by identifying the top-k highest ranked items for a

certain user. It is not necessary to learn the rating values in order to offer suggestions using the later technique. These systems are referred to as Top-N recommendation systems. The term ”item” applies to the suggestions made by the algorithm to users. A RS often specializes on one category of item that it recommends. It may be videos on an online platform, travel destinations on a tour guide, medication to patients or links on a social network. Prediction receivers are often customers of an online platform, but they can also be businesses or even virtual agents. The mechanism by which ratings are captured influences the design of recommender systems. Ratings are sometimes expressed on a scale that reflects the degree of like or dislike for the object in question.

System	Recommended Items
Amazon.com	Books and other products
Netflix	Streaming Videos
GroupLens	News
GoogleNews	News
Youtube	Online videos
TripAdvisor	Travel products
IMDB	Movies

Figure 2.1: Real systems and the recommendations they make

Figure 2.1 summarizes some common applications of recommender systems and their goals. Many of them belong to the field of e-commerce. The topic of recommender systems increased applicability to online businesses is examined in [13]. Nevertheless, recommender systems have evolved further the typical realm of product suggestions. To boost the growth of their network, online social platforms usually recommend links to their customers. An example of a social networking website is Facebook. While in an e-commerce scenario recommendation systems are utilized to directly increase profits by recommending items, in these applications this is achieved in a less obvious way. Increasing the number of social connections improves a user’s experience on a social network, so the network’s advertising revenue largely depends on it.

2.1.1 Recommender Systems Categorization

The following section discusses the main types of RS. Based on the input data, the models of recommender systems can be labeled as follows: collaborative filtering (CF), content-based systems, knowledge-based systems, and hybrid systems.

The collaborative filtering [14, 15] approach is based on collecting and analyzing user-item interaction data. Let us take the example of a movie recommendation system, where a movie is recommended to you because it received a high rating from a user with similar interests as you. The interaction data (ratings) are presented in the form of a utility matrix [16, Chapter 9]. Figure 2.2 provides an example of this concept. In the matrix representation, users are arranged in rows and items in columns, and for each user-item combination, one cell contains the value of the rating. The concept of collaborative filtering is to exploit the correlation that often exists between users and items to predict the gaps in the utility matrix. Depending on how the predictions are made, CF methods are described as Memory-Based or Model-Based.

	Movie 1	Movie 2	Movie 3	...	Movie M
User 1	2	5	1	...	3
User 2	0	1	4	...	0
..	...	...	...	...	...
User N	0	0	2	...	1

Figure 2.2: Example of a NxM Utility Matrix in a movie recommender system

Memory-Based methods the user's neighborhood to provide recommendations. These methods can be further divided into:

- User-Based CF [17]: ratings are predicted by like-minded users. First, a similarity function is applied among users and the k-nearest neighbors for each user are stored. We can then recommend the most popular items among the k nearest neighbors.
- Item-Based CF [18]: predicts ratings by finding items that are similar to the ones that the user has already gave positive feedback. We compute a similarity function between

the user's "favorite" item and all other items. We keep k-nearest neighbors that are not observed by the user and recommend those items.

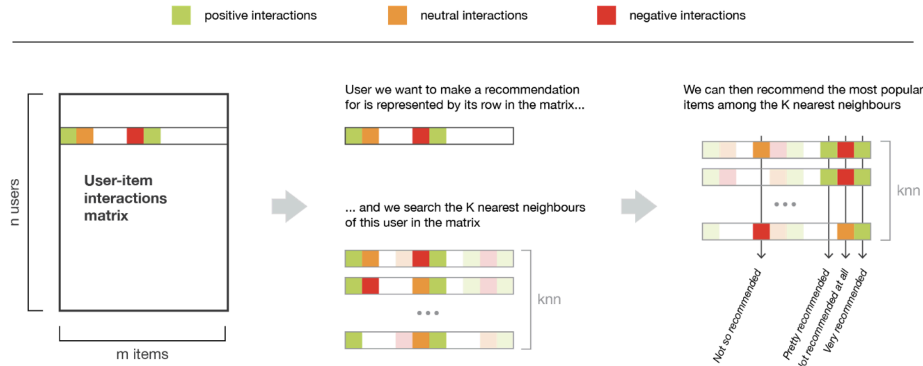


Figure 2.3: Illustration of User-Based CF method

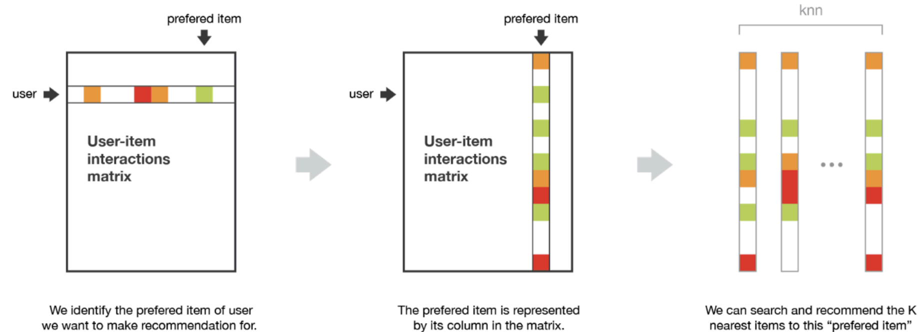


Figure 2.4: Illustration of Item-Based CF method

Model-Based methods develop models for user ratings to provide recommendations. The modeling process is performed using various ML methods such as decision trees, rule-based models, and Bayesian methods. According to Su and Khoshgoftaar [19], this approach is more powerful than memory-based systems and can also handle sparsity issues better.

In Content-Based systems [20, 21], user ratings and history are combined with item attributes to make recommendations. Although the content-based methods give good recommendations for new items, they are not able to perform well for new users because of the absence of user's historical ratings. Researchers refer to this problem as the Cold-Start [22].

When features are modeled according to user-specific constraints, such as filters or search fields, etc., we speak of knowledge-based systems [23]. These systems are particularly useful in cold-start environments where limited data is available. Depending on user input,

knowledge-based methods can be constraint-based [24] - the user specifies ranges and value filters - or case-based [25, 26], i.e., the user specifies example items and the system finds similar items.

To be more versatile, recommender systems require specific methods. It is interesting to note that different types of systems can work for different types of data. If we consider a scenario where different types of data are available, we can combine aspects of the above methods to achieve more effective results. These systems are referred to as hybrid systems[27].

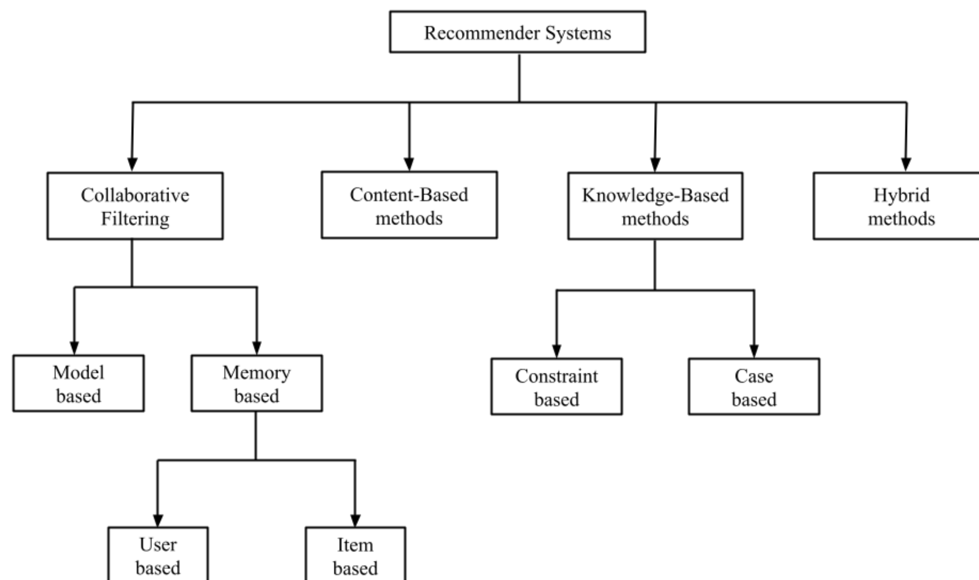


Figure 2.5: Categories of different RS methods

2.1.2 Evaluation

In online evaluation methods [28] real users participate in the recommendation task. It is often difficult to gain access to such systems, and even when it is, they are usually designed for a specific environment. When evaluating algorithms, it may be useful to test multiple data types to verify the generalizability of the proposed solution. All these reasons seem to confirm that online systems are not ideal for benchmarking or research. On the other hand, offline evaluation methods train and evaluate on historical data sets. Some notable recorded datasets commonly used in RS research are: MovieLens, Netflix Prize Dataset, and Amazon

Review Data. We must acknowledge that offline methods are the most popular techniques because they are fast, inexpensive, and can often be evaluated on a variety of datasets.

One of the most widely used metrics for comparing algorithms is accuracy. In the most general case, accuracy represents the percentage of user ratings correctly predicted by the algorithm, similar to regression modeling. Although the Accuracy metric is one of the most important aspects of evaluation, it occasionally provides an insufficient overview of the user experience. Instead, other secondary goals such as coverage, novelty, serendipity, diversity and robustness can provide useful insights into performance. Coverage is a measure of the proportion of total items that the algorithm was able to recommend. Novelty measures the probability that a recommender system makes recommendations that the user has not seen before. The authors of [29] explain the concept of novelty. Serendipity quantifies the proportion of unexpected recommendations that only an non-obvious recommender would provide. Diversity measures how diverse the set of recommended items is. High diversity can ensure better novelty and serendipity. Robustness and stability are measures of the system's vulnerability to attack (e.g., fake reviews). A study that looks at the reasons and motives why some users enter fake ratings can be found in [30]. Scalability is also an important metric as the size of data sets increases. It measures the performance of recommender systems in terms of training time, prediction time, and memory usage.

An standard measure of ranking quality commonly used in studies is the Normalized Discounted Cumulative Gain (NDCG). In a recommendation scenario, the algorithm recommends a set of items $P = \{p_1, p_2, ..., p_N\}$ to a user u. Items are associated with a relevance score. Then, the cumulative gain(CG) comes of the graded sum of the relevance scores of all the items in the list.

$$CG_p = \sum_{i=1}^{p} rel_i$$

In Discounted Cumulative Gain, relevance scores are weighted by the logarithm of the position at which they appear. This is done because DCG assumes that items with high relevance scores are more useful if they appear early in the recommendation list. We can calculate DCG as follows

$$DCG_u(P) = \sum_{i=1}^{N} \frac{relevance_u(p_i)}{\log_2(i+1)}$$

It is then normalized by the overall relevance of each item by computing an ideal DCG, the

IDCG.

$$NDCG_u(P) = \frac{DCG_u(P)}{IDCG_u(P)} \tag{2.1}$$

The value of NDCG is in the range (0,1) and is often reported in research as NDCG@k. The slate size k, see Chapter 3, is the number of items recommended to the user each time.

2.2 Deep Learning

Deep learning is a field of machine learning based on artificial neural networks(ANNs), an architecture heavily influenced by human biology. Deep neural networks have excelled in a number of areas and have been responsible for most of the significant recent advances in machine learning and AI. They are critical components of some of the most exciting technologies, such as self-driving cars [31], image recognition systems [32] and speech recognition systems [33]. One of the most remarkable aspects of ANNs that has led to their success is the ability to train on vast volumes of data and extract features without supervision.

The building unit of ANNs is called an artificial neuron. A neuron expects an input vector $x = [x_0, x_1, x_2, ..., x_n]$ and computes the weighted sum of the inputs with respect to the weights w. Then, a bias b is added to the weighted sum, which is used to shift the activation function up or down. The output of the neuron is passed to the activation function f to output the final result, that can be mathematically framed as:

$$y_i = f(\sum_j x_j w_{i,j} + b_i)$$

Without the use of an activation function, the output will always be linear. Activation functions introduce non-linearity in ANNs, which allows them to detect complicated underlying patterns in data. Some famous activation functions are:

- Binary Step Function:

$$f(x) = \begin{cases} 0 & \text{if} x < 0 \\ 1 & \text{if} x \geq 0 \end{cases}$$

- Sigmoid:

$$f(x) = \frac{1}{1 + e^{-x}}$$

- Hyperbolic Tangent:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- Rectified Linear Unit(ReLu):

$$f(x) = \max(0, x)$$

Feed-forward networks are the simplest form of ANN. Neurons are organized into layers, which can be of three types: Input, Hidden, and Output. This structure is illustrated in Figure 2.7. As the name implies, information travels in only one direction, from the input layer to the output layer. The term ”deep” refers to the number of hidden layers in its architecture. A single-layer neural network is called a Perceptron, Figure 2.6.

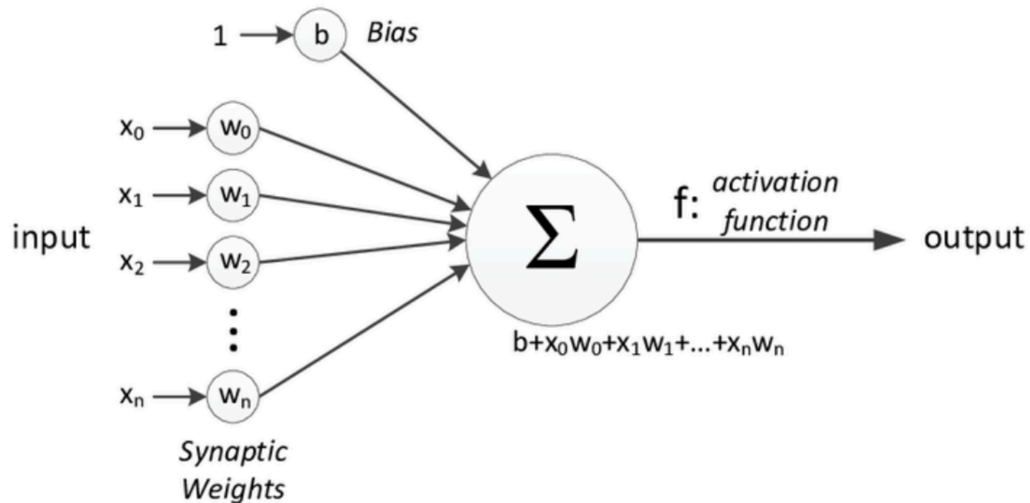


Figure 2.6: Perceptron

Training is an iterative process in which neural network parameters (weights and biases) are gradually adjusted with the goal of minimizing a loss function, which is often a summary of errors during training. Once the loss is computed, the information is back propagated layer by layer until each neuron receives a signal (gradient) describing its contribution to the loss. This technique is called backpropagation. The optimizer is the module that tells us in which direction to alter the parameters of the network and is usually based on gradient descent optimization. Some improved variants of stochastic gradient descent, such as ADAM [34] and AdaGrad [35], have also been proposed. The learning rate is the amount by which the optimizer progresses in the direction of the gradient.

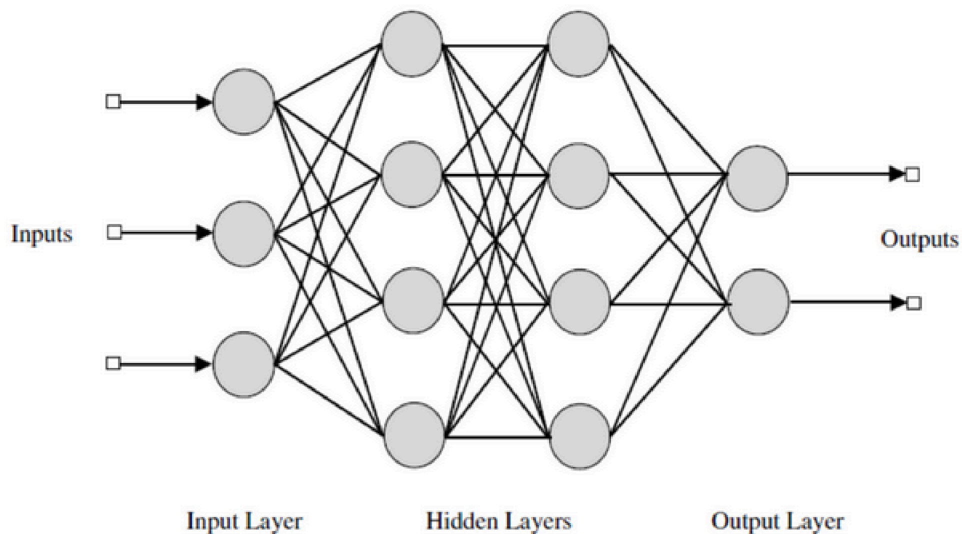


Figure 2.7: Feed Forward Network

2.3 Reinforcement Learning

This section introduces Reinforcement Learning, following the formalism from one of the most influential textbooks in the field by Suton et al [36]. In contrast to supervised learning, where algorithms are trained using labeled data. Reinforcement Learning is a branch ML algorithms where an agent interacts with an environment and learns the solution to a problem by trial and error. The problem is simulated in a discrete timeline in an RL setting. At each time step t, the agent is in a state $s_t \in S$ and interacts with the environment by taking an action $a_t \in A(s)$. Then, the agent receives a new observation of the environment and moves to a new state $s_t + 1$. In addition, the agent receives a reward signal indicating how good the performed action was for that state.

This dynamic system can be formulated as a Markov Decision Process(MDP). The term MDP derives from the fact that the system follows the Markov property, which states that transitions are based only on the most recent state and action, without past historical data influencing the choice. The MDP is a 5-tuple $M = \{S, A, R, P, \gamma\}$ where:

- S is a set of all the states, called state space
- A a set of all the available actions, called action space
- P a set of state transition probabilities, describing the probability to update to a state

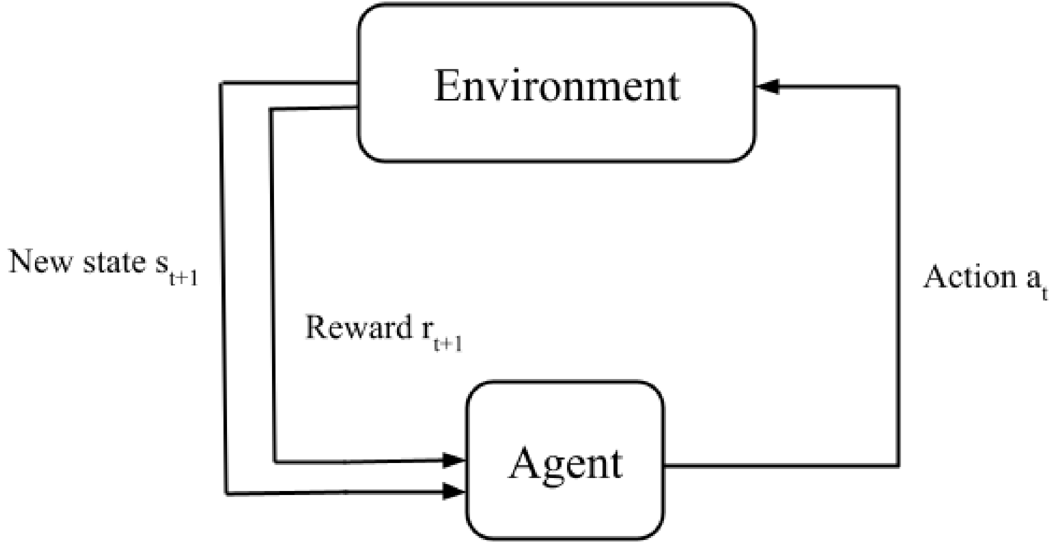


Figure 2.8: Agent-environment interaction loop

s' from a state s, under an action a

- R a reward signal obtained from transition from state s to s' due to an action a
- a discounted factor $\gamma \in (0, 1]$

A policy is a mapping from states to actions in an MDP, a plan that tells the agent what to do. Policies might be deterministic, $\pi(s)$, based only on state s, or stochastic, i.e., defining a probability distribution over actions based on state, $\pi(a \mid s)$. The goal of an RL agent is to select a policy that optimizes the predicted sum of rewards, known as the return G_t, and written as follows:

$$G_t = r_t + r_{t+1} + r_{t+1} + ... + r_{N-1} \tag{2.2}$$

where N represents the length of an episode and t the time step. For continuous tasks, Eq 2.2 is formulated as follows:

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{2.3}$$

The value function estimates the expected sum of rewards the agent will receive if it pursues a given policy in a given state. It is a measure of how good state s is for the agent. Assuming a state s and a policy π are selected, the resulting value $v_\pi(s)$ in that state is expressed mathematically as in 2.4.

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s],\ for\ all\ s\ \in\ S \tag{2.4}$$

The recursive form of Equation 2.4 is called the Bellman equation and is shown in Equation 2.5.

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \textstyle\sum_\alpha \pi(\alpha \mid s) \sum_{s'} P^\alpha_{s,s'}[R^\alpha_{s,s'} + \gamma \mathbb{E}_\pi[G_t \mid S_{t+1} = s']] \\ &= \textstyle\sum_\alpha \pi(\alpha \mid s) \sum_{s'} P^\alpha_{s,s'}[R^\alpha_{s,s'} + \gamma v_\pi(s')] \end{aligned} \tag{2.5}$$

Similarly, the action value function, often known as Q function, estimates the expected total of rewards the agent receives when choosing an action a in state s according to a policy.

$$\begin{aligned} q_\pi(s,a) &= \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[\textstyle\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a],\ for\ all\ s\ \in\ S\ and\ a\ \in\ A \end{aligned} \tag{2.6}$$

The objective of an RL agent is to optimize a policy $\pi*$. Traditional RL algorithms accomplish this by storing each pair of state values in an array. These methods are called tabular methods and can be further divided into Dynamic Programming, Monte Carlo methods, and Temporal Difference (TD).

Dynamic Programming approaches search for appropriate policies based on modeling the environment and a value function. Policy iteration and value iteration are two significant algorithms from this class. Monte Carlo methods, on the other hand, do not require complete knowledge of the environment. They are able to learn from experience through simulated or real scenarios. TD Methods are a combination of the previous ones. They learn from interaction with the environment, without a model as in Monte Carlo, and perform updates to current estimates as in DP. Popular algorithms in this category of methods are Q-Learning and SARSA.

2.4 Deep Reinforcement Learning

Traditional RL methods work well for simple problems when the state/action space is small enough to accommodate a Q-table in memory. However, in real applications, the state space is usually vast, which limits the size of the table due to memory constraints in hardware. In addition, tabular methods model the state space with discrete variables only. To overcome

the aforementioned restrictions, it is common to estimate the value function or a policy using a Deep Neural Network.

2.4.1 Deep Q Network

Deep reinforcement learning really exploded in 2015 with the development of the Deep Q-Learning algorithm [37], by the DeepMind team at Google. This method is known as DQN because it models the value of choosing each action a in a given state s as the output of a neuron with parameters θ. For example, Q-learning learns the parameters by repeatedly minimizing a series of loss functions and explicitly approximates the ideal value function $Q(s, a) \approx Q(s, a; \theta)$

One of the key innovations of this algorithm that has contributed to its popularity is the use of a replay buffer [38]. The replay buffer solves a very fundamental problem in Deep Reinforcement Learning. The problem is that real networks tend to produce erroneous results when their inputs are correlated. What can be more correlated than an agent playing a game where each time step depends on the one immediately preceding it? These correlations cause the agent to exhibit very strange behavior: It knows how to play the game and suddenly forgets when a new set of states occurs that it has never seen before. The neural network cannot really generalize from previously seen states to unseen states because of the complexity of the parameter space of the underlying problems. The replay buffer fixes this problem by allowing the agent to randomly sample transitions from many different episodes. This ensures that these time steps are completely uncorrelated. The agent thus gets a broad sample of the parameter space and can therefore learn a more robust policy with respect to new inputs.

2.4.2 Policy Gradients

A limitation of DQN is that computing the value for each action requires one neuron, so the action space must be discrete. If the action space is continuous or stochastic, a policy-gradient approach is suitable. Policy gradient methods [39] aim to optimize a policy $\pi(\alpha \mid s, \theta)$ directly, rather than using a value function, by formulating the output as a probability distribution of expected returns for each action separately. They can also be applied to discrete spaces by post-processing the output with softmax layer.

The quality of the policy is measured by its performance $J(\theta)$. The objective function in

Equation 2.7 aims to maximize the scalar value $J(\theta)$.

$$\theta^* = \underset{\theta}{\operatorname{argmax}} J(\theta) \tag{2.7}$$

Since a deep neural network is used to approximate the policy, the parameter θ is the neural network weights. The updates of θ are done via the gradient ascent (Equation 2.8)

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\theta_t) \tag{2.8}$$

2.4.3 Actor Critic

Actor Critic techniques were developed as a fusion of earlier models. In the actor critic techniques, two neural networks are used. One of them is utilized to estimate the agent's policy directly. The critic network, on the other hand, is used to predict the value function. The critic tells the actor how good each action is based on whether the ensuing state is useful. The two networks cooperate to determine how to act effectively in the environment. The actor selects actions, the critic analyzes the states, and the output is matched against the rewards of the environment. The critic grows increasingly precise in approximating the state values with time and allows the actor to choose the actions that contribute to those states. Figure 2.9 illustrates the core concept underlying this approach. In practice, the weights of the two deep neural networks are adjusted at each time step based on the TD error.

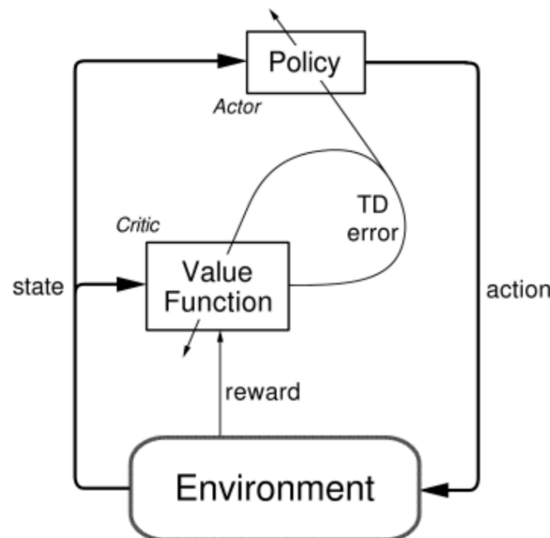


Figure 2.9: Actor-Critic architecture

2.5 Current Work

Recently, interest in Deep Learning methods has grown in many areas such as speed recognition and computer vision. The main reason for this is the ability of deep neural networks to learn complex features from scratch, while producing state-of-the-art results. Recommender systems are no exception. The number of publications from the RS community using Deep Learning techniques has grown exponentially. However, a detailed discussion of Deep Learning based RS is beyond the scope of this thesis. For a detailed overview see [40].

In this section, we give a high level overview of some of the most cited RL-based recommender systems. The work in [41] is a well-written survey that discusses in more detail the methods used so far.

2.5.1 RL-based Recommender Systems

Web Watcher [42] is the very first application of RL in recommender systems. Web Watcher is a bot on the School of Computer Science website at CMU. Similar to a travel guide, its goal is to allow people to quickly discover content that matches their interests. They use RL, and in particular Q-learning, to model and solve the recommendation problem. Since then, Q-learning has been very popular in RS field [43, 44, 45, 46, 47, 48, 49].

However, traditional Reinforcement Learning methods have not been very successful in optimizing recommender systems. In practice, most RL agents operating in the recommendation environment focus on optimizing the user's immediate response to the recommended item. In other words, they are very greedy and optimize for short-term rewards. In most toy situations this is adequete, but in recommender systems there are no trivial tradeoffs between the user's short-term and long-term response. The reason that most RL attempts have ignored these long-term effects is that they are notoriously difficult to model in typical recommender environments because there are an incredible number of users and items to recommend. As a result, exploration of this action space is almost always intractable.

This is the problem that the Slate-Q algorithm [50] attempts to solve. The novelty of this approach is the utility of two assumptions. One is related to rewards and the other is related to state, which simplifies the learning computation and allows us to efficiently estimate the long-term value of each items in the system. Researchers have empirically shown that the Slate Q algorithm outperforms traditional Q learning on this difficult task in both average

rewards per episode and click-through rate. This research sheds light on exciting applications of reinforcement learning and recommender systems, and paves the way for training efficient, scalable agents using domain-specific, model-based approaches.

Next, we review DRN [51]. In this work, researchers address the challenges that exist in news recommendation. They employ a DQN agent to capture the dynamic nature of news domain. They also define user activeness, a new metric besides traditional click/no-click that can model how often users return to their platform, potentially having a positive impact on accuracy. Finally, they demonstrate the performance of their system through offline and online evaluation, using the DQN agent in a real-world news recommendation application.

Finally, there is another approach: DARES [3]. The researchers have developed a decentralized framework for distributed asynchronous training, borrowing some concepts from Federated Learning. Their architecture is based on the A3C algorithm and consists of a set of local workers interacting with their copy of the environment and a global model that updates its parameters based on the gradients computed by the workers. The main difference with the original A3C algorithm is that the workers of DARES keep their data local. To simulate the recommendation task, they use PyRecGym [52], which was developed by the authors in a previous work. Their results are comparable to the state of the art. Finally, they show a significant performance improvement in terms of time complexity by using multiple parallel workers without any significant compromise in accuracy.

2.5.2 Simulation Environments

The notion of an RL simulation environment has gained prominence as RL has evolved, offering the research community accessibility to a reliable environment for exploring, testing, and designing RL agents and methodologies. A simulation environment defines RL agents' present state, provides potential actions, and determines the rewards for those actions. Environments are often developed considering a particular RL task so that they may be customized to meet the goals of the problem. Depending on the optimization problem posed, this can be a tedious task [53]. Several open source projects offer standardized methods for constructing RL agents and environments and for training and gathering the results of the algorithms and policies, even if some of the standard source code still needs to be written. The OpenAI Gym [54] is among the most prominent RL toolkits. It specializes in the application of RL in gaming contexts.

In terms of gym environments for RS evaluation, recent approach, RecSim [1], proposes a platform that enables the development of configurable environments and RL agents. RecSim conceptually simulates how a recommendation agent might interact with a user model, a document (item) model, and a user choice model environment. The agent can access observable attributes of simulated individual users and documents to make suggestions to users by suggesting lists of documents (called slates). RecSim provides some already developed environments.

Another RL gym specifically designed for RS research is the PyRecGym [52]. This work focuses on popular open-sourcing datasets (MovieLens, Outbrain, Adressa), but also provides a general, customizable data pipeline for converting any dataset. It works with various input data types and has multiple RL agents to begin. The data is split into initialization data and interaction data. The former is used to train the user simulators, while the latter is used to train the agents. To simulate user responses to actions and the resulting observations and rewards, the gym implements a reward and observation loop.

A different gym environment worth reviewing is MarsGym [55]. It simulates the dynamics of a marketplace. In this work, a more holistic approach was employed to enhance the satisfaction of all parties participating in the marketplace (users, suppliers, third parties) and thus the health and fairness of this environment. In this work, Fairness measures are offered to guarantee that the models respect satisfaction and awareness for all affected parties, as well as to identify any bias in vulnerable attributes.

Chapter 3

Simulation Environment

In this Chapter, we present our approach to simulating the environment for the recommendation problem. We use ML Fairness Gym, which builds on the previously introduced RecSim environment. We explain the building blocks of a RecSim environment in more detail. We also present the dataset and techniques used to create feature vectors and simulate ratings.

3.1 Dataset

Before proceeding with the method and modeling, we would like to introduce the dataset we based our approach on. We worked on a movie recommendation system and trained the agents on the popular MovieLens 1M dataset[1] [56].

The GroupLens research lab in the Department of Computer Science and Engineering at the University of Minnesota first created the MovieLens dataset in 1997. On MovieLens, users rated the movies they watched on a 5-star scale and added their own tags to characterise them. The version used in this paper contains 1,000,209 anonymous ratings of about 3883 movies given by 6,040 MovieLens users who joined MovieLens in 2000. One advantage of the MovieLens dataset is that it is widely used in the field of recommender systems in both academia and industry. This allows us to easily reproduce the results and compare them with state-of-the-art algorithms. The dataset consists of three files: users.dat, movies.dat and ratings.dat.

[1] Url to download MovieLens 1M dataset: `http://files.grouplens.org/datasets/movielens/ml-1m.zip`

All ratings are in the “ratings.dat” file in the format:

$$UserID :: MovieID :: Rating :: Timestamp$$

- User IDs range between 1 and 6040.
- Movie IDs range between 1 and 3883
- Ratings are one a 5-star scale, with whole ratings only
- Timestamp is presented in seconds using Unix time

User information, including some demographic data provided by the users, are include in the “users.dat” file in the format:

$$UserID :: Gender :: Age :: Occupation :: Zip-code$$

- User IDs are mapped to a linear index, to optimize speed.
- Genre is denoted as “M” for male, or “F” for female
- Age is represented in groups, from the following ranges:

1: “Under 18”	18: “18-24”	25: “25-34”
35: “35-44”	45: “45-49”	50: “50-55”
56: “56+”		

- Occupation is selected from Table 3.1

Finally, “movies.dat” file includes movie information and follows the format:

$$MovieID :: Title :: Genres$$

3.2 Formulation of recommendation in RL

In this section, we explain how a Deep Reinforcement Learning-based recommender system was developed to mimic the distributed training scenario of the A2C/A3C algorithms. We simulate the recommendation task as a sequential decision problem, where the recommender (i.e., the agent) interacts with users (i.e., the environment) to gradually offer a list of

0: “other or not specified”	1:“academic/educator”
2: “artist”	3: “clerical/admin”
4: “college/grad student”	5: “customer service”
6: “doctor/health care”	7: “executive/managerial”
8: “farmer”	9: “homemaker”
10: “ K-12 student”	11: “lawyer”
12: “programmer”	13: “retired”
14: “sales/marketing”	15: “scientist”
16: “self-employed”	17: “technician/engineer”
18: “tradesman/craftsman”	19: “unemployed”
20: “writer”	

Table 3.1: Choices of occupations in the dataset

things at different time steps. Our RL agent aims to maximize the cumulative rewards of the entire recommendation process. More specifically, we model the recommendation process through an MDP, described in section 2.3 as follows.

Actions: Prior studies have formulated actions in many ways, including binary actions, such as recommending or not recommending an item, or multiclass actions, such as rating predictions on a 5-star scale. However, in this environment, the action space includes all 3883 movies that can be recommended at any given time. The actions can be either a single movie or a list of movies called a slate. A slate is defined as a subset of all items accessible to the user.

States: As in RecSim environments, the state space is represented as a concatenation of user features, the user’s response history, and the features of the target item (see Table 3.2).

During feature extraction, a one-hot encoding vector is created for the genre feature vector of the most recently evaluated movie. One-hot encoding is one of the most classic encoding techniques. A single categorical data value is converted into a list of Boolean values, one for each category, indicating which category the data point falls under. In addition, each movie is associated with a violence score extracted from the Tag Genome dataset [57]. This dataset contains the tag relevance scores that make up the tag genome. The tag relevance represents the relevance of a tag to a movie on a continuous scale from 0 to 1.

Rewards: After the agent performs an action in a state, the environment (i.e., the user)

provides feedback, and based on this feedback, a reward is given to the agent. The user's response contains two features, which are the simulated rating and the violence score. The reward signal is in the range $(0, 1)$ is mathematically expressed as:

$$R_t = \frac{1}{N}\sum_{i=1}^{N} \lambda(1 - violence_score_i) + (1 - \lambda)rating_i$$

where N denotes the size of the slate. We encode the violence score and λ weight into the reward, so that recommending too many violent movies brings a negative reward. The embeddings used to generate the reward are not observable by the agent, as this is partly what the RL agent tries to estimate. More details on how environment simulates the ratings are presented in section 3.3.

Table 3.2: Features of the MoviesLens dataset used

Feature Type	MovieLens Dataset
User Features	
Category	age gender occupation postcode
Item Features	
Integer	rating violence score
Text	genre

3.3 Simulation Environment

In this section, the simulation environment used is presented in more details. When comparing simulation environments, some desirable features of interest are the use of large data sets, the availability of real world data, detailed documentation, and the time required to simulate each time step. In a similar work in [3], PyRecGym was adapted to simulate the recommendation task. Unfortunately, an open source version of PyRecGym was not available.

Since none of the other environments studied were suitable for this application, an alternative approach was to develop our own custom environment.

For the simulation task, we use ML Fairness Gym [58], which uses RecSim as a backend. We extend the user model code, described in section 3.3.1, to include the user features of the MovieLens dataset mentioned above. Then we develop a wrapper that follows the API of OpenAI Gym framework. The resulting simulation environment contains three important methods.

Initialization: It initializes and loads all feature vectors for each user and object. It also sets up the state space and action space of the environment.

Reset It resets the internal state and action space of the environment and prepares it to start a new episode.

Step: Performs a step through the environment. It takes an action chosen by the agent and returns a list that includes an observation of the current state, a reward signal, a Boolean variable that indicates reaching a final state, and a dictionary of additional information about the environment.

Finally, in ML Fairness Gym we also added two important evaluation metrics to measure the performance of our algorithm: Click-Through-Rate and NDCG, see Chapter 5.

3.3.1 RecSim simulation

RecSim requires the configuration of 4 levels of abstraction to simulate specific features of user behavior, depicted in Figure 3.1. The environment consists of a User model, a Document model, a User Choice model and a User-Transition model.

A User Model defines the logic of user generation. It describes which features represent the users in the environment and how they are sampled: from a distribution or from a dataset. In this thesis, the user model is configured with the demographic features of the users from the MoviesLens dataset, such as gender, age, occupation, and zip code.

Similarly, a Document Model is responsible for generating documents (items) and selecting features to represent them in the environment. In our case, items include features from the items of the MovieLens dataset such as title, genre, year, and the violence score they are associated with. When the agent recommends a document to the user, the user's response is simulated by a User-Choice model.

A User-Choice model has access to all user features as well as observable document

features. Latent document features, such as the subject or the quality of the document, can also influence the user's response. In the proposed environment, the information is derived via a matrix factorization of MovieLens ratings.

Once the user-document interaction occurs, a User-Transition model is used to update the user state. For example, a user's interest may increase or decrease after interacting with a particular item category. RecSim influences the behavior of users after they have interacted with certain movie categories by encoding the addiction dynamics into the user embedding obtained by matrix factorization.

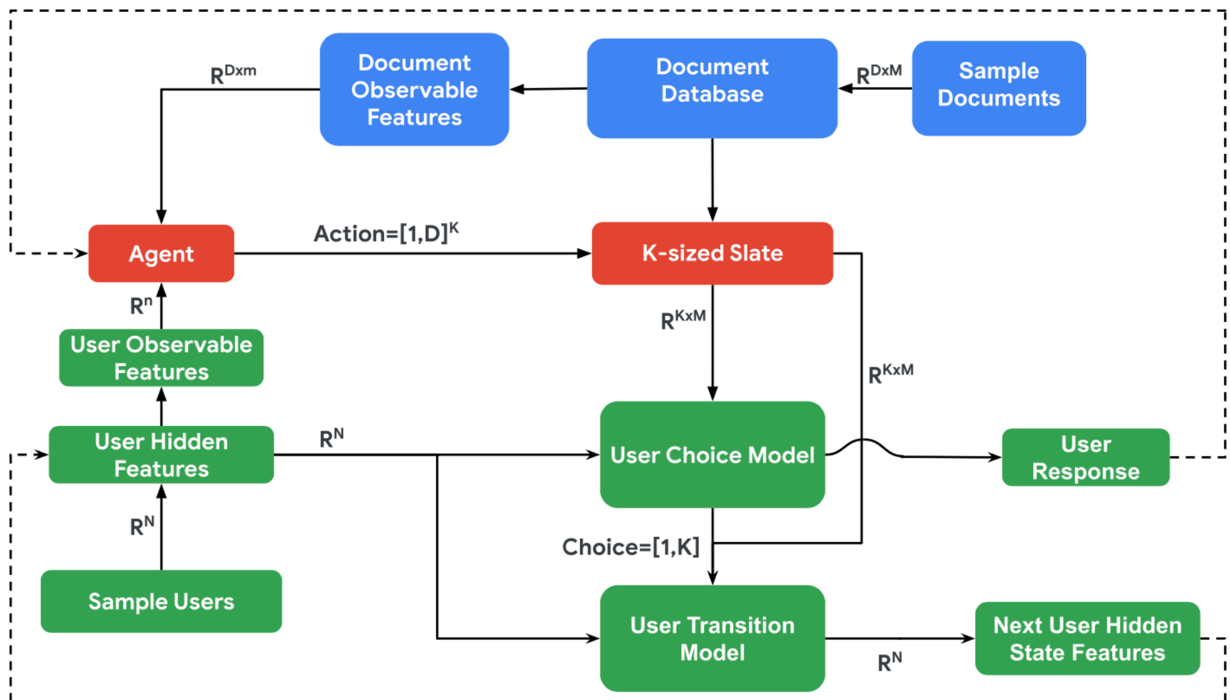


Figure 3.1: Data Flow through components of RecSim [1]

3.3.2 How ratings are simulated

User ratings are stored in the form of a utility matrix, $R^{m \times n}$, as described in Figure 3.2, where $m = 6040$ is the number of users and $n = 3706$ the number of movies. The problem is that the matrix R is often high-dimensional and sparse, since we do not assume that each user has seen and rated all movies. Our goal is to predict (simulate) the unseen ratings, i.e., the cells in the matrix with 0 values that represent movies that users have not seen. We approach the rating prediction problem using matrix factorization. This technique assumes that we can approximate our matrix R as the dot product of two low-dimensional matrices $R^{m \times n} \approx$

$U^{m \times k} \times M^{k \times n}$, Figure 3.3.

```
utility_matrix
```

movieId	0	1	2	3	4	5	6	7	8	9	...	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882
userId																					
0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6035	0.0	0.0	0.0	2.0	0.0	3.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6036	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6037	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6038	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6039	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

6040 rows × 3706 columns

Figure 3.2: User- movie interaction matrix from MovieLens dataset

We use a non-negative SVD model to compute the matrices U and M that contain latent user features (user embeddings) and movie latent features (movie embeddings), respectively. We call them latent features because they arise from the underlining patterns of the data. They are not based on human-defined features such as comedy, action, etc. In our case, $k = 55$ is the number of latent features. The higher k is, the more accurate the predictions are. After factors are computed, we can predict the rating r_{ij}, that user i gave to movie j by:

$$r_{ij} = u_i^T m_j$$

This type of prediction is actually a form of compression. Once these features are learned, we can discard the original data we started with because we can simply multiply the features together to recreate that data. In terms of memory, we first need to store $6040 \times 3706 = 20572240$ interactions. After matrix factorization, the total number of parameters reduces to $6040 \times 55 + 3706 \times 55 = 536030$. However this technique is limited due to the usage of embeddings. We can't add new users or movies unless we retrain the model.

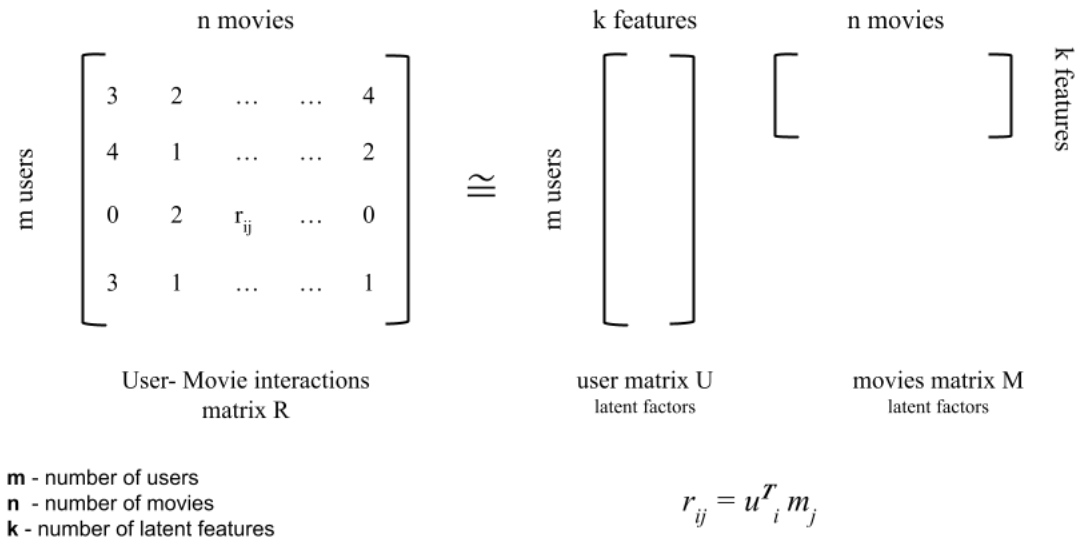


Figure 3.3: User- movie interaction matrix from MovieLens dataset

Chapter 4

Methods

This chapter introduces the Advantage Actor-Critic algorithms and the neural networks architecture. We present the asynchronous learning process, and we discuss the implementation details of our code.

4.1 Asynchronous Training using Distributed Agents

The high computational efficiency of the A3C algorithm originates from its use of separate learner agents executed in parallel, Figure 4.1. The proposed system consists of two main elements: The central neural network model (master) and the local learner agents (workers). Using a stochastic gradient descent-based optimization technique, these agents asynchronously update the parameters of the central neural network model. Each worker receives a local copy of the global neural network model. The local DNN model computes each state's losses and the gradients by interacting with their environment instance in parallel. At each episode termination, the central network model is updated asynchronously using the parameters of each learner agent's network instance. Before an agent starts a new episode, its local copy is overwritten with the newly trained global model. Figure 4.2 provides an overview of the training process that each worker goes through. The A3C approach is an ensemble of several basic reinforcement learning agents, comparable to DQN. Their combined impact on the central network determines how well it functions. The training data becomes more varied as the learner agents can freely explore their environments. A3C is multi-threaded; splitting the main training operation over several CPU cores is simple.

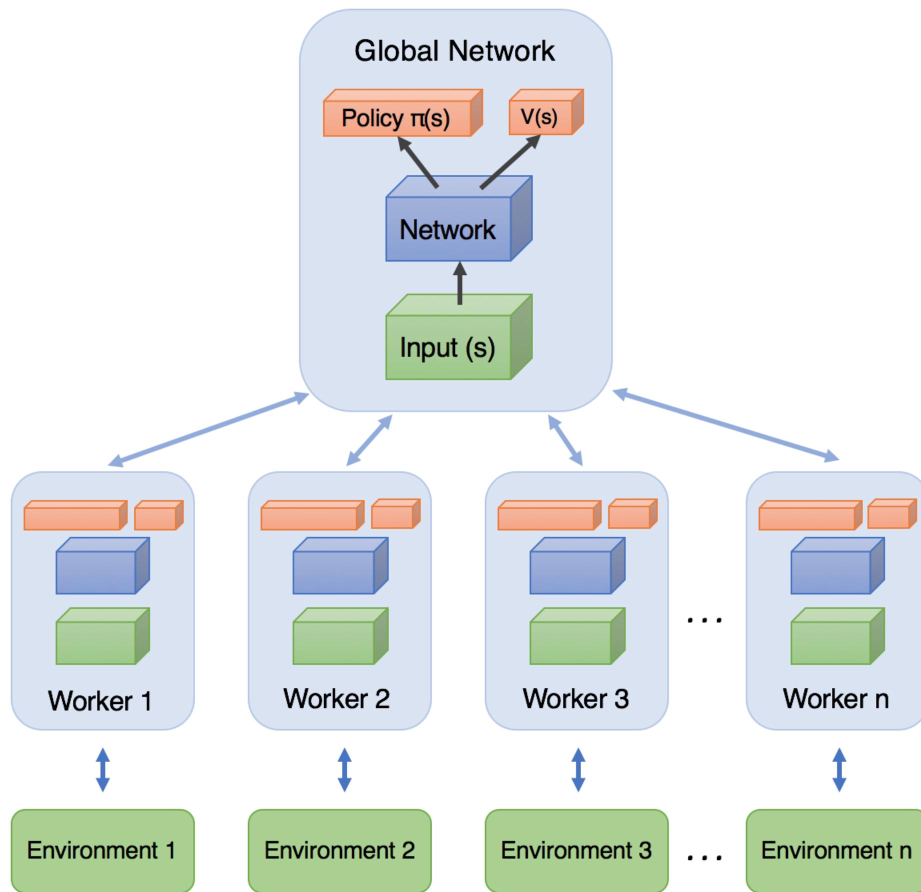


Figure 4.1: A3C high level architecture

4.2 Actor-Critic and Advantage

Reviewing the concepts from section 2.4.3, Actor-Critic Methods keep track of both a policy (i.e., the Actor) that directs how the agent acts and an estimate of the value function (i.e., the Critic) that assesses how effective action is. In A3C, the Q values are calculated directly. Instead, R_t is used to estimate $Q(s_t, \alpha_t)$. The Actor is represented by the policy $\pi(\alpha_t|s_t;\theta)$, while the Critic from the advantage function $A(s,\alpha;\theta)$ formulated by

$$A(s_t, \alpha_t; \theta) = R_t(s_t, \alpha_t) - V(s_t; \theta)$$

Ideally, the value of the advantage function will converge to zero because this will means that the model estimates a state value equal to the received reward. The parameterization of π and V can be done using neural networks. We implement the algorithm using a single

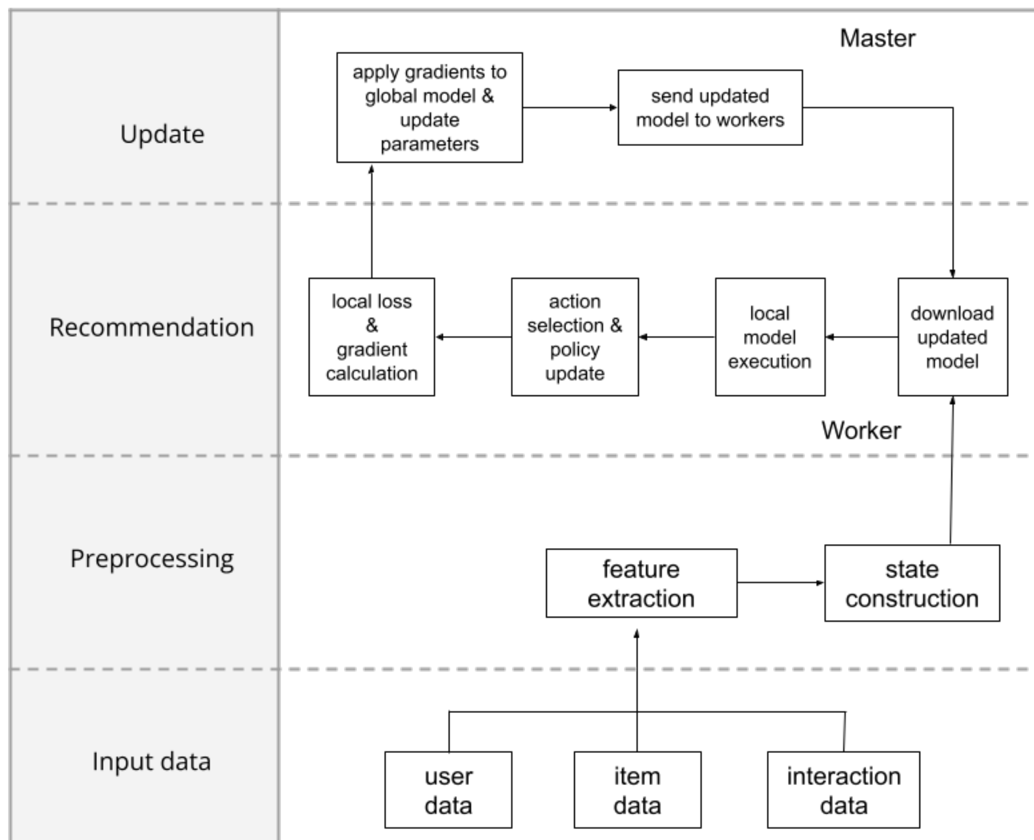


Figure 4.2: Advantage Actor-Critic process

neural network with two hidden shared layers. The Actor and Critic networks diverge from the last shared hidden layer into their dedicated layers. The output of the Actor network is a Softmax network, while the Critic network ends with a single neuron layer, providing the value estimate. The network architecture is illustrated in Figure 4.3.

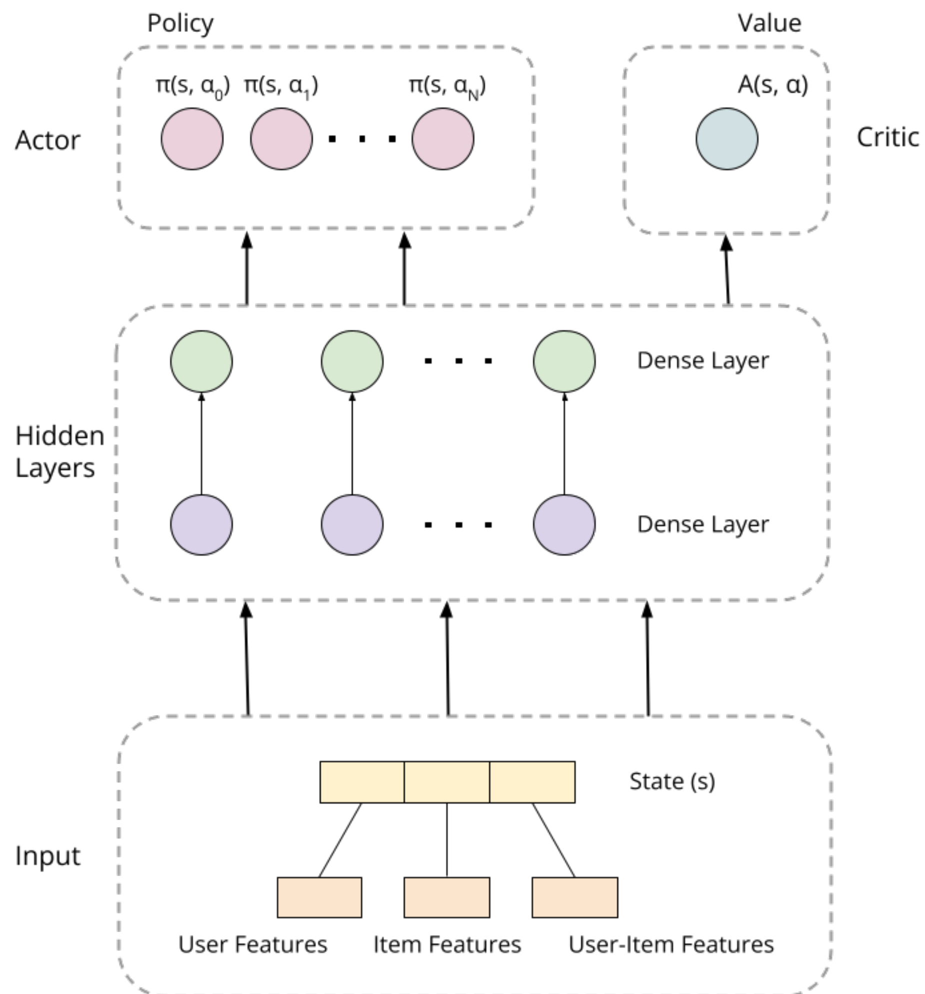


Figure 4.3: The design of Actor-Critic neural network

4.3 Evaluation Agents

Two different agents will be used to assess the effectiveness of our RL agents: our implementation of the A3C agent and the A2C agent from the open-source library stable-baselines3 [59]. This library provides a set of reliable implementations of Reinforcement Learning agents.

Algorithm 1 Asynchronous Advantage Actor-Critic

1: Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$
2: Assume thread-specific parameter vectors θ' and θ'_v
3: Initialize local learner agent step counter $t \leftarrow 1$
4: **repeat**
5: Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$
6: Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
7: $t_{start} = t$
8: Get state s_t
9: **repeat**
10: Perform action α_t according to policy $\pi(\alpha_t|s_t : \theta)$
11: Receive reward r_t and new state s_{t+1}
12: $t \leftarrow t + 1$
13: $T \leftarrow T + 1$
14: **until** terminal s_t or $t - t_{start} == t_{max}$
15: $R = \begin{cases} 0 & \text{for terminal state } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal state } s_t \end{cases}$
16: **for** $i \in \{t_{t-1}, ..., t_{start}\}$ **do**
17: $R \leftarrow r_i + \gamma R$
18: Accumulate gradients w.r.t. θ' : $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(\alpha_i|s_i; \theta')(R - V(s_i; \theta'_v))$
19: Accumulate gradients w.r.t. θ'_v : $d\theta_v \leftarrow d\theta_v + \frac{\partial(R - V(s_i; \theta'_v))^2}{\partial \theta'_v}$
20: **end for**
21: Perform asynchronous update of θ using $d\theta$ and θ_v using $d\theta_v$
22: **until** $T \geq T_{max}$

Figure 4.4: Advantage Actor-Critic algorithm from [2]

Figure 4.4 presents the algorithmic formulation of A3C for each parallel learner agent, summarizing the sections above. It should be noted that the agents train the global neural

network using the RMSProp optimizer through the lock-free Hogwild paradigm [60].

A2C is a synchronous deterministic version of the A3C algorithm. In contrast to A3C, when an agent finishes a training episode waits for the others. Then, the global model averages over all the workers and updates its parameters. One problem arising from this is that some workers may be idle most of the time, which wastes resources.

4.3.1 Implementation details

The source code developed and used for the experiments in this thesis is publicly stored in a GitHub repository[1]. All code is written in Python using TensorFlow [61]. For the deep neural network implementation, we used Keras [62], a deep learning API built on top of TensorFlow. Our implementation of A3C is based on the official GitHub repository of TensorFlow [63]. The directory structure of the repository is shown in Figure 4.5. The file structure is similar to the one recommended by the OpenAI API. The *MovieLensEnv* class wraps an OpenAI Gym environment as ML Fairness Gym Environment. The ML Fairness Gym is forked in the repository and contains the changes we made to extend its code, see Chapter 3. The agents/A2C directory contains all the code to run experiments with the A2C agent from stable-baselines3. Our implementation of A3C can be found under the agents/A3C directory. A diagram illustrating the classes for the environment and the agents can be found in Figure 4.6. The main functions of these classes are explained in the following sections.

The *ActorCriticModel* class defines the neural network architecture that implements both the policy (actor) and value (critic) approximator models for A3C. As we mentioned earlier, in our implementation the policy and value networks share their hidden layers. The output of the actor-critic model in a forward pass are the logits and values. The logits are post-processed by a *Softmax* layer to convert them into a discrete probability distribution over the actions. The *ActorCriticModel* class inherits from the *keras.Model* class, allowing easy configuration and inference of the neural networks without having to implement them from scratch.

The *MasterThread* class is the starting point for the A3C agent. It creates an instance copy of an *ActorCriticModel* model in which the trainable parameters of the global model are stored. It also implements the *train()* method, the main function of the training loop, which initializes and creates a number of workers. The number of workers is limited to the CPU threads on the local machine. Finally, it also contains an *eval()* method for testing the

[1]GitHub repository: `https://github.com/vstergiou/thesis-recsys`

```
thesis_recsys/
    mlfairnessgym/
    data/
    gymrecsys/
        envs/
            movielens_env.py
        agents/
            A2C/
            A3C/
                actorcritic_model.py
                a3c_master.py
                a3c_worker.py
                memory.py
```

Figure 4.5: Repository file structure

agent.

The *WorkerThread* class extends the Python *Threading.Thread* class. It overrides the run() method. This is the default function called when a worker thread is started by the Master thread. Each worker keeps a local copy of the environment it interacts with and a local *ActorCriticModel* instance. It computes the policy and value networks and updates the global model accordingly. Workers are implemented as threads, so that they can share memory. This type of execution allows the global parameters to be shared directly without the need to apply more complex techniques as in a multiprocessor scenario.

Finally, the *Memory* class is used to store the agent's state, action, rewards and the evaluation metrics that will later be used to plot the learning statistics.

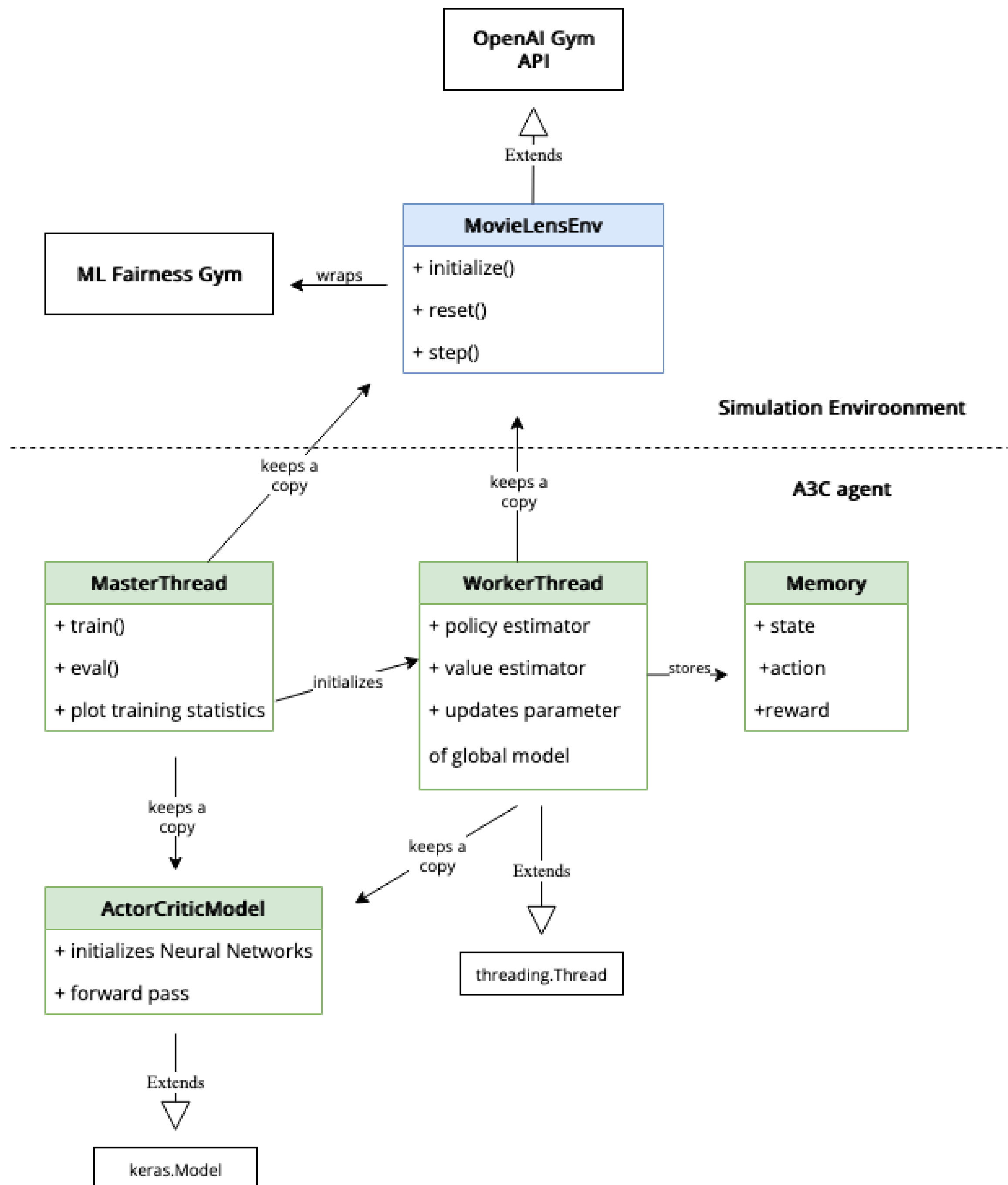


Figure 4.6: Class diagram

Chapter 5

Results

In this chapter, we aim to demonstrate the performance of the A2C and A3C agents. We compare the agents to similar work from other researchers and evaluate them using the metrics CTR and NDCG. Finally, we investigate whether and how the number of workers affects the training time of the algorithms, and draw conclusions that give us guidance for more optimal implementations.

5.1 Evaluation Methods

Implementing an extensive search in the hyper parameter space would not be efficient because of the enormous number of network parameters models have. Instead, we use the hyper parameters proposed in the original paper.

Since A3C is asynchronous, we follow a similar evaluation procedure as in the original paper [2]. We run each experiment 5 times for 2000 episodes and average the results. These results should be a fair representation of the actual performance of the algorithm. To better understand the agents' performance, we add a random recommendation baseline as a comparison. Finally, the best model is used to run an experiment with a slate size of 10 and compute the NDCG metric.

All experiments are performed locally on a MacBook Pro with the M1 chip with 8 CPU cores, and one thread per CPU core. The number of workers is set to 8.

5.2 Training Results

This section analyzes the results obtained from the various experiments we performed using the implemented simulation environment. The first experiment aims to set a random agent as a baseline. Each episode has a length of 50, which means that the agent recommends to the user 50 items. Since the reward signal is normalized in the range (0,1), the maximum cumulative reward for each episode is set to 50. In an ideal scenario, users rated each movie that was recommended to them a 5. The total reward for this episode would then be 50. Given this and the results of Figure 5.1, it stands to reason that the random agent had an average rating of about 3.2/5 during the last episode. A comparison of Figure 5.1 and Figure 5.2 reveals that the original Actor Critic method did not perform better than the Random Agent. In the next experiment, we investigate whether more sophisticated Actor-Critic methods produce better results.

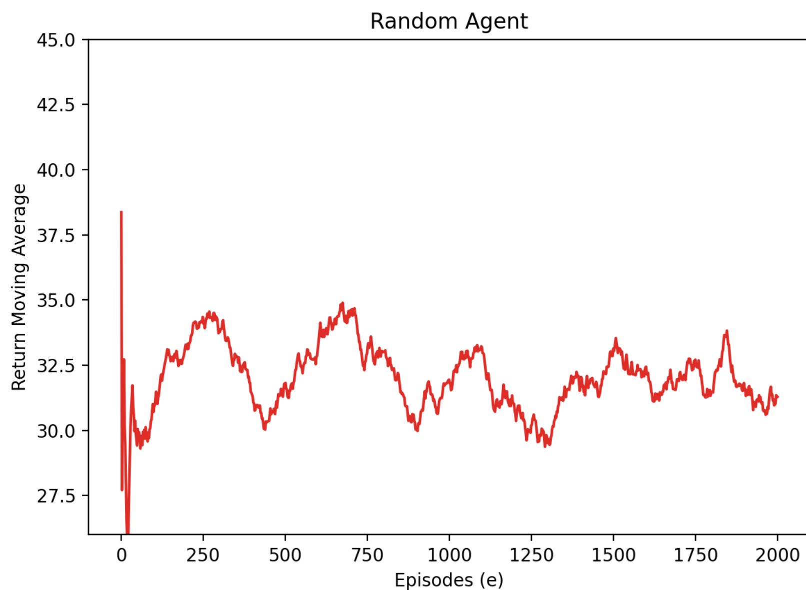


Figure 5.1: Random Agent

Figure 5.3 summarizes the results of training both agents A2C and A3C. The red horizontal line represents the random baseline. From this data, we can see a significant improvement in user satisfaction. In the last training episode, A2C and A3C had achieved user ratings of 4.1/5 and 3.9/5, respectively. The results also indicate that training for more episodes could potentially improve performance. Additionally, these models didn't undergo any hyper parameter tuning.

At this point, it must be taken into account that the agents initially have no knowledge

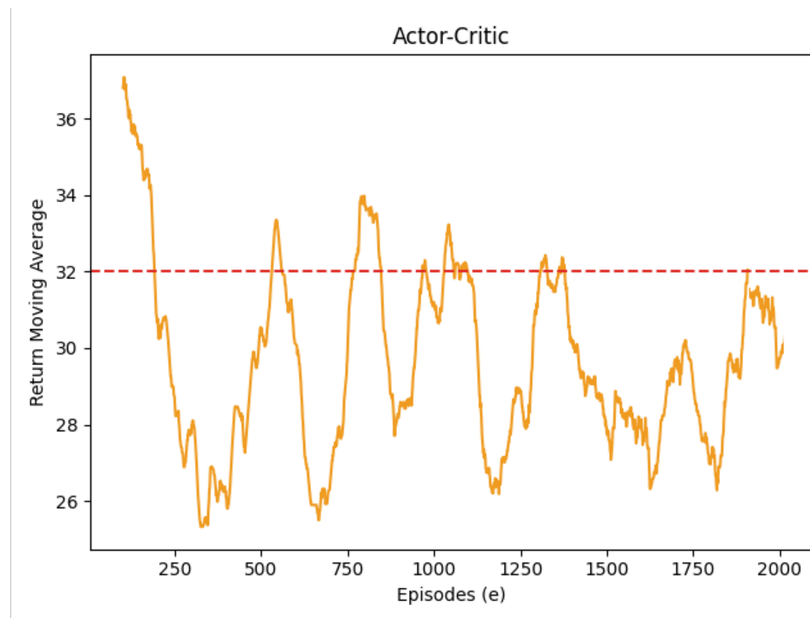


Figure 5.2: Actor-Critic Agent

of the environment. While it could yield better results, addressing the Cold Start problem is beyond the scope of this thesis. The results reveal that A2C outperforms the A3C algorithm. However, further analysis of Figure 5.3 shows that A3C deliver better results in the first episodes. We assume this is due to the asynchronous nature of A3C; the training data becomes more diverse, resulting in less bias. However, it can be argued that A3C does not provide better recommendations than A2C in the long run.

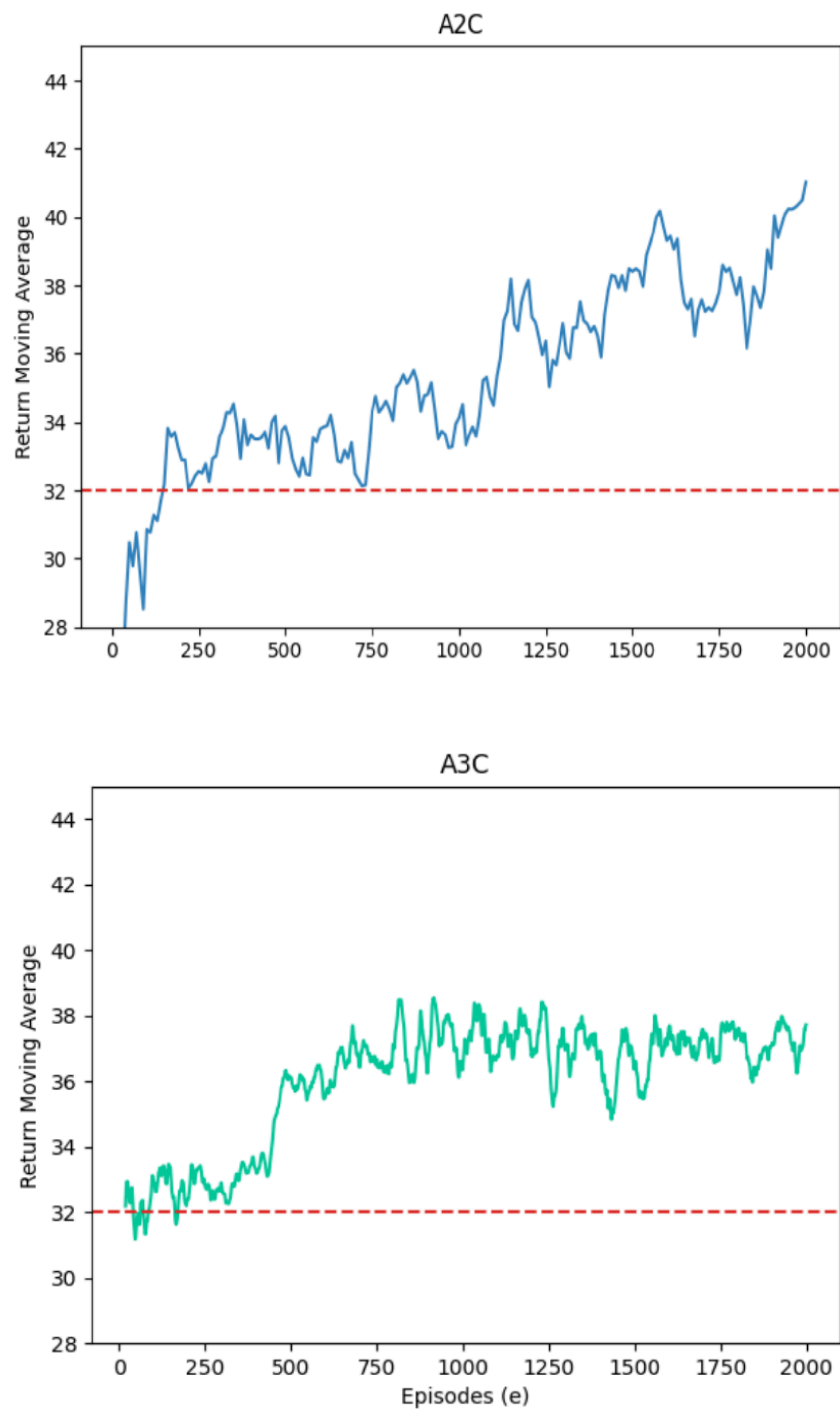


Figure 5.3: Comparison of A2C and A3C agents

To further compare the performance of these two agents with each other and with other RL agents in the literature, we included the Click-Through-Rate (CTR) metric in our implementation. Since our recommendation problem is formulated with 5-star ratings and not in a binary setting (click/no click), we model positive feedback (click) as a $rating > 3.5$.

As shown in Figure 5.4, A2C achieves higher performance CTR. The A2C agent reported an increase of about 6% CTR. The global average CTR for A2C was 0.606, and 0.54 for A3C. Based on the patterns of the previous experiments, it is evident that the A2C agent is trained to make better recommendations and achieve both higher CTR rates and average rewards.

Next, we will compare our agents with other implementations. The authors of DARES did a comparison of their system with some basic agents in Reference [3]. They trained the models on the MovieLens100k dataset for 20000 episodes and used CTR as the reward. MovieLens100k is a smaller version of the dataset we used. The results of their analysis can be summarized in Table 5.3. The global average CTR of the models used in the comparison is in the range (0.719, 0.882), which is approximately a 12-28% increase in performance. This is a significant difference for the performance of CTR. As Rajabi notes in [64], "a small improvement in the predictive accuracy of CTR can generate millions of dollars in revenue for the advertising industry".

What is interesting about the data in the table is that the reported performances were

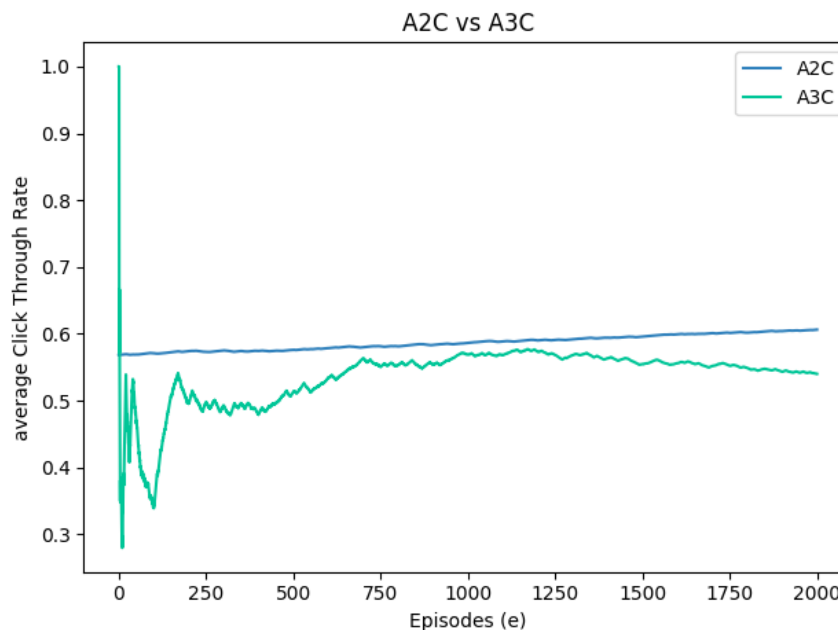


Figure 5.4: Comparison of A2C and A3C agents

Dataset	Model	CTR	Reference
MovieLens100k	LinUCB	0.78	[65]
	DQN	0.747	[66]
	DDQN	0.747	[66]
	A2C-F	0.741	[67]
	A2C-D	0.719	[67]
	DARES	0.882	[3]
MovieLens1M	**A2C**	**0.606**	This thesis
	A3C	**0.5402**	This thesis

Table 5.1: Comparison of our implemented agents with the models presented in [3]

achieved after 20000 training episodes, whereas our models were trained for 2000. This was mainly the result of limited computational power on the local computer and the large number of experiments we had to run. To investigate this further, we analyze the graph in Figure 5.5, which shows the average CTR during episodic training of the referenced models. We can see that the A2C-D algorithm, which is similar to our A2C, has an average CTR of about 0.67 for 2000 episodes, which is closer to our results. Finally, we note in Figure 5.4 that CTR of the A2C increases linearly, suggesting that further training could increase performance and possibly provide more comparable results.

5.3 Evaluation Results

In this section, we evaluate our best model (A2C) with k =10 as the size of the slate and the number of items recommended to the user each time. We recommend movies from the test set to the user and calculate the global average CTR at the end of the evaluation process. Similar to the previous section, we want to compare our results with the mentioned models. We recommend movies from the test set to the user and calculate the global average CTR at the end of the evaluation process. The results in Table 5.2 show that DARES outperforms the other methods. It is very interesting to note that while the rest of the methods had higher training scores CTR, our method produced comparable and sometimes better results in the evaluation. This can be explained by the selection of the dataset. MovieLens1M is quite similar to MovieLens100k, but users have more interactions with movies on average. This

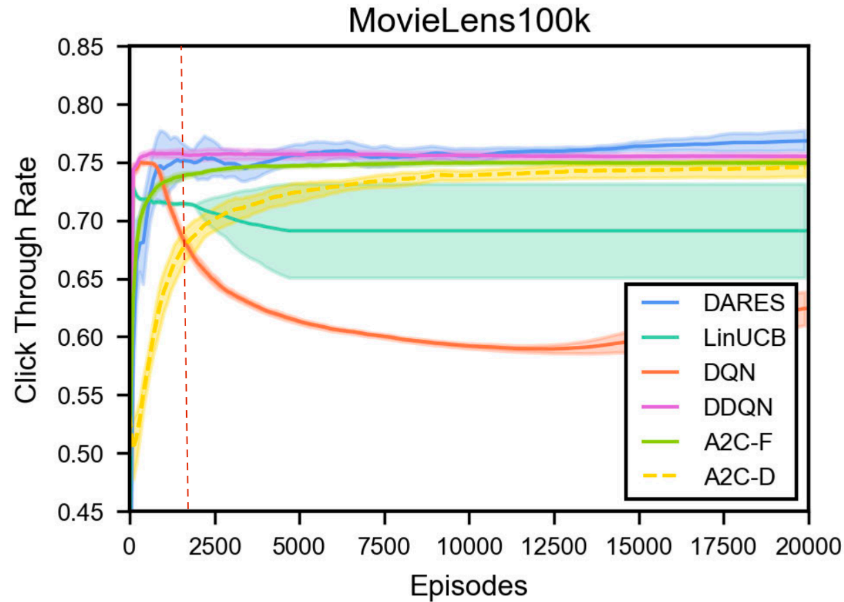


Figure 5.5: Training statistics illustrating the average CTR per episode for the various models compared in DARES [3]

benefits our method when the size of the recommendation slate increases.

Dataset	Model	CTR	Reference
MovieLens100k	LinUCB	0.268	[65]
	DQN	0.227	[66]
	DDQN	0.227	[66]
	A2C-F	0.220	[67]
	A2C-D	0.199	[67]
	DARES	0.454	[3]
MovieLens1M	**A2C**	**0.231**	This thesis

Table 5.2: Comparison of our implemented agents with the models presented in [3]

5.3.1 Relevance score

Click-Though-Rate is a commonly used metric for evaluating the performance of an algorithm. However, as pointed out in the paper in [68], ”algorithms with higher overall CTR do not necessarily correspond to higher relevance.” To better understand our agent’s ability to provide good recommendations, we used NDCG as a relevance metric. We test the model

with the best performance (A2C) using the NDCG@10 metric (Eq. 2.1). We conduct experiments with slates. In this scenario, the agent recommends a list of 10 items to the user. The results of the evaluation analysis are shown in Table 5.3.

	NDCG@10 A2C	NDCG@10 Random
Mean	0.8172	0.6714
Min	0.4174	0.2968
Max	0.9723	0.8432

Table 5.3: Comparison of evaluation metrics for A2C agent against random

It is evident that the A2C agent performs better than random. Nevertheless, we should not be surprised, as was implied from the experimental results of the previous section. We define relevance as the simulated 5-star rating. Thus, the ideal relevance is a rating of 5, which makes an ideal slate of items one in which all the recommended movies received a 5/5 rating.

5.4 Number of workers and training time

In this set of experiments, we analyze how the number of workers affects the training time. We test algorithms A2C and A3C for different numbers of workers from this list [1,2,4,8,16]. For each number of workers, we ran the experiments with 3 different seeds and the results were averaged.

Table 5.6 illustrates the impact of different numbers of workers on A3C training time. Originally, we would expect the number of training steps per second to increase linearly due to the asynchronous nature of the A3C algorithm. However, we observe the opposite effect. As the number of workers increases, the number of training steps per second decreases. This can be justified by the threading nature of our implementation. In Python, the Global Interpeter Lock doesn't allow a process to execute multiple threads in parallel. Therefore, in our case, updates to the global model occur concurrently, which can cause bottlenecks as the number of workers increases. A more optimal and fully asynchronous solution would be to implement the workers as distinct processes.

From the data in Figure 3, we can see that the A2C agent leads to slower training times than the A3C agent. This is the main drawback of this algorithm. As workers synchronously update the global model, many of them become idle and wait for the others to finish their

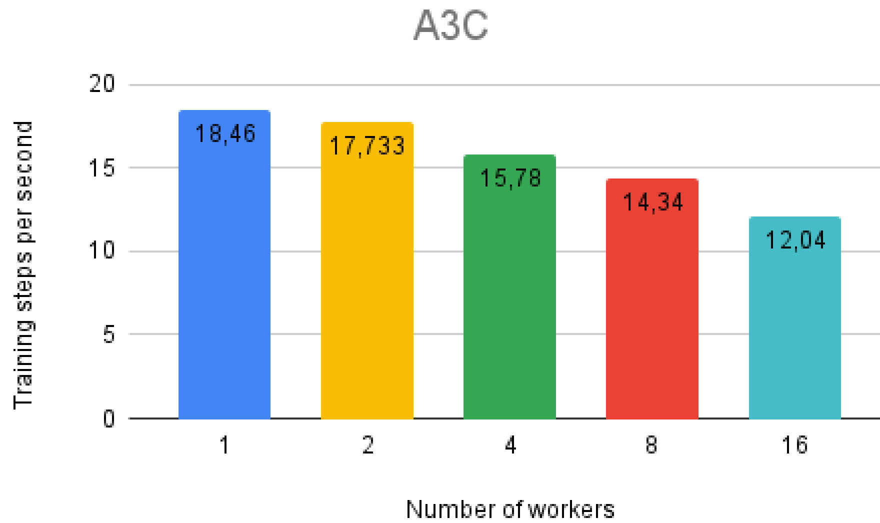


Figure 5.6: A3C: Number of workers vs training steps per second.

episode. Given this, A3C is usually faster due to the communication cost of the subprocesses. An optimal solution that A2C could benefit from is to use a GPU with large batch sizes.

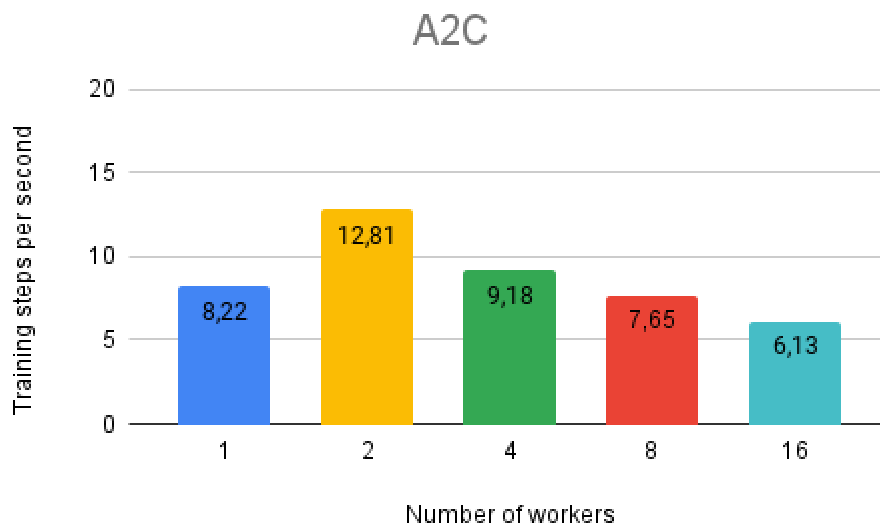


Figure 5.7: A2C: Number of workers vs training steps per second.

Chapter 6

Conclusions

In this thesis, we designed, implemented and evaluated a distributed recommender system using Deep Reinforcement Learning and based on the asynchronous Advantage Actor-Critic algorithm. A thorough investigation was conducted to identify the available RL environments that are suitable for realistic settings of recommender systems. This was a challenging task as it was found that the open source implementations of such environments are sparse, not actively maintained, and poorly documented. It is evident that researchers need to shift their focus in this area to further accelerate progress.

Next, different RL agents were trained on the selected environment. Initially, the vanilla Actor-Critic method failed to produce better results than random. A possible direction for future work is to investigate how the performance of this agent is affected by using the Adam optimization method.

However, in our experiments with the more sophisticated Advantage Actor-Critic agents A2C and A3C, significant differences in user satisfaction were found. The models were compared using the average cumulative rewards during the last training episode. We also compared these algorithms to agents from related studies using the global average CTR. The results of the experiment showed that our agents performed lower but may be able to achieve comparable results. In addition, the NDCG was used as an evaluation measure to show that the A2C agent provides better recommendations than the random agent. Finally, we conducted some experiments to investigate the effect of the number of parallel workers on the training time and analyzed the results.

A very interesting path for future research is to use other RS metrics. Despite the importance of accuracy metrics, it is not always possible to measure the true effectiveness of

recommender systems in real applications. Some examples of metrics that could be used are novelty, serendipity, and coverage.

Bibliography

[1] Eugene Ie, Chih-Wei Hsu, Martin Mladenov, Vihan Jain, Sanmit Narvekar, Jing Wang, Rui Wu, and Craig Boutilier. Recsim: A configurable simulation platform for recommender systems. *CoRR*, abs/1909.04847, 2019.

[2] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1928–1937. JMLR.org, 2016.

[3] Bichen Shi, Elias Z. Tragos, Makbule Gulcin Ozsoy, Ruihai Dong, Neil Hurley, Barry Smyth, and Aonghus Lawlor. Dares: An asynchronous distributed recommender system using deep reinforcement learning. *IEEE Access*, 9:83340–83354, 2021.

[4] Sheena Iyengar and Mark Lepper. When choice is demotivating: Can one desire too much of a good thing? *Journal of personality and social psychology*, 79:995–1006, 01 2001.

[5] Francisco García-Sánchez, Ricardo Colomo Palacios, and Rafael Valencia-García. A social-semantic recommender system for advertisements. *Inf. Process. Manag.*, 57(2):102153, 2020.

[6] Thi Ngoc Trang Tran, Alexander Felfernig, Christoph Trattner, and Andreas Holzinger. Recommender systems in the healthcare domain: state-of-the-art and research issues. *J. Intell. Inf. Syst.*, 57(1):171–201, 2021.

[7] Peter Chapman, Janet Clinton, Randy Kerber, Tom Khabaza, Thomas P. Reinartz, Colin Shearer, and Richard Wirth. Crisp-dm 1.0: Step-by-step data mining guide. 2000.

[8] Charu C. Aggarwal. *Recommender Systems - The Textbook*. Springer, 2016.

[9] Mohit Sewak. *Deep Reinforcement Learning - Frontiers of Artificial Intelligence*. Springer, 2019.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[11] David Goldberg, David A. Nichols, Brian M. Oki, and Douglas B. Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35(12):61–70, 1992.

[12] Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors, *Recommender Systems Handbook*, pages 1–35. Springer, 2011.

[13] J. Ben Schafer, Joseph A. Konstan, and John Riedl. Recommender systems in e-commerce. In Stuart I. Feldman and Michael P. Wellman, editors, *Proceedings of the First ACM Conference on Electronic Commerce (EC-99), Denver, CO, USA, November 3-5, 1999*, pages 158–166. ACM, 1999.

[14] John S. Breese, David Heckerman, and Carl Myers Kadie. Empirical analysis of predictive algorithms for collaborative filtering. *CoRR*, abs/1301.7363, 2013.

[15] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In John B. Smith, F. Donelson Smith, and Thomas W. Malone, editors, *CSCW '94, Proceedings of the Conference on Computer Supported Cooperative Work, Chapel Hill, NC, USA, October 22-26, 1994*, pages 175–186. ACM, 1994.

[16] Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman. *Mining of Massive Datasets, 2nd Ed.* Cambridge University Press, 2014.

[17] Liu Na, Ming-Xia Li, Qiu Hai-yang, and Hao-Long Su. A hybrid user-based collaborative filtering algorithm with topic model. *Appl. Intell.*, 51(11):7946–7959, 2021.

[18] Wentao Zhao, Huanhuan Tian, Yan Wu, Ziheng Cui, and Tingting Feng. A new item-based collaborative filtering algorithm to improve the accuracy of prediction in sparse data. *Int. J. Comput. Intell. Syst.*, 15(1):15, 2022.

[19] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009, jan 2009.

[20] Soumen Chakrabarti. *Mining the web - discovering knowledge from hypertext data*. Morgan Kaufmann, 2003.

[21] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008.

[22] Xuan Nhat Lam, Thuc Vu, Trong Duc Le, and Anh Duc Duong. Addressing cold-start problem in recommendation systems. In Won Kim and Hyung-Jin Choi, editors, *Proceedings of the 2nd International Conference on Ubiquitous Information Management and Communication, ICUIMC 2008, Suwon, Korea, January 31 - February 01, 2008*, pages 208–211. ACM, 2008.

[23] R. Burke. Knowledge-based recommender systems. 2000.

[24] Alexander Felfernig and Robin D. Burke. Constraint-based recommender systems: technologies and research issues. In Dieter Fensel and Hannes Werthner, editors, *Proceedings of the 10th International Conference on Electronic Commerce 2008, Innsbruck, Austria, August 19-22, 2008*, volume 342 of *ACM International Conference Proceeding Series*, pages 3:1–3:10. ACM, 2008.

[25] Derek G. Bridge, Mehmet H. Göker, Lorraine McGinty, and Barry Smyth. Case-based recommender systems. *Knowl. Eng. Rev.*, 20(3):315–320, 2005.

[26] Fabiana Lorenzi and Francesco Ricci. Case-based recommender systems: A unifying view. In Bamshad Mobasher and Sarabjot S. Anand, editors, *Intelligent Techniques for Web Personalization, IJCAI 2003 Workshop, ITWP 2003, Acapulco, Mexico, August 11, 2003, Revised Selected Papers*, volume 3169 of *Lecture Notes in Computer Science*, pages 89–113. Springer, 2003.

[27] Robin D. Burke. Hybrid recommender systems: Survey and experiments. *User Model. User Adapt. Interact.*, 12(4):331–370, 2002.

[28] Tariq Mahmood and Francesco Ricci. Improving recommender systems with adaptive conversational strategies. In Ciro Cattuto, Giancarlo Ruffo, and Filippo Menczer, ed-

itors, *HYPERTEXT 2009, Proceedings of the 20th ACM Conference on Hypertext and Hypermedia, Torino, Italy, June 29 - July 1, 2009*, pages 73–82. ACM, 2009.

[29] Joseph A. Konstan, Sean M. McNee, Cai-Nicolas Ziegler, Roberto Torres, Nishikant Kapoor, and John Riedl. Lessons on applying automated recommender systems to information-seeking tasks. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*, pages 1630–1633. AAAI Press, 2006.

[30] Shyong K. Lam and John Riedl. Shilling recommender systems for fun and profit. In Stuart I. Feldman, Mike Uretsky, Marc Najork, and Craig E. Wills, editors, *Proceedings of the 13th international conference on World Wide Web, WWW 2004, New York, NY, USA, May 17-20, 2004*, pages 393–402. ACM, 2004.

[31] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

[32] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[33] Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Process. Mag.*, 29(6):82–97, 2012.

[34] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[35] John C. Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, 2011.

[36] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998.

[37] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nat.*, 518(7540):529–533, 2015.

[38] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[39] Richard S. Sutton, David A. McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller, editors, *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, pages 1057–1063. The MIT Press, 1999.

[40] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Comput. Surv.*, 52(1):5:1–5:38, 2019.

[41] Mohammad Mehdi Afsar, Trafford Crump, and Behrouz H. Far. Reinforcement learning based recommender systems: A survey. *CoRR*, abs/2101.06286, 2021.

[42] Thorsten Joachims, Dayne Freitag, and Tom M. Mitchell. Web watcher: A tour guide for the world wide web. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan, August 23-29, 1997, 2 Volumes*, pages 770–777. Morgan Kaufmann, 1997.

[43] Anongnart Srivihok and Pisit Sukonmanee. E-commerce intelligent agent: personalization travel support agent using Q learning. In Qi Li and Ting-Peng Liang, editors, *Proceedings of the 7th International Conference on Electronic Commerce, ICEC 2005, Xi'an, China, August 15-17, 2005*, volume 113 of *ACM International Conference Proceeding Series*, pages 287–292. ACM, 2005.

[44] Pornthep Rojanavasu, Phaitoon Srinil, and Ouen Pinngern. New recommendation system using reinforcement learning. 01 2005.

[45] Nima Taghipour, Ahmad Kardan, and Saeed Shiry Ghidary. Usage-based web recommendations: A reinforcement learning approach. In *Proceedings of the 2007 ACM Conference on Recommender Systems*, RecSys '07, page 113–120, New York, NY, USA, 2007. Association for Computing Machinery.

[46] Nima Taghipour and Ahmad A. Kardan. A hybrid web recommender system based on q-learning. In Roger L. Wainwright and Hisham Haddad, editors, *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC), Fortaleza, Ceara, Brazil, March 16-20, 2008*, pages 1164–1168. ACM, 2008.

[47] Tariq Mahmood, Ghulam Mujtaba, and Adriano Venturini. Dynamic personalization in conversational recommender systems. *Information Systems and e-Business Management*, 12, 05 2013.

[48] Binbin Hu, Chuan Shi, and Jian Liu. Playlist recommendation based on reinforcement learning. In Zhongzhi Shi, Ben Goertzel, and Jiali Feng, editors, *Intelligence Science I - Second IFIP TC 12 International Conference, ICIS 2017, Shanghai, China, October 25-28, 2017, Proceedings*, volume 510 of *IFIP Advances in Information and Communication Technology*, pages 172–182. Springer, 2017.

[49] Jia-Wei Chang, Ching-Yi Chiou, Jia-Yi Liao, Ying-Kai Hung, Chien-Che Huang, Kuan-Cheng Lin, and Ying-Hung Pu. Music recommender using deep embedding-based features and behavior-based reinforcement learning. *Multim. Tools Appl.*, 80(26):34037–34064, 2021.

[50] Eugene Ie, Vihan Jain, Jing Wang, Sanmit Narvekar, Ritesh Agarwal, Rui Wu, Heng-Tze Cheng, Tushar Chandra, and Craig Boutilier. Slateq: A tractable decomposition for reinforcement learning with recommendation sets. In *Proceedings of the Twenty-eighth International Joint Conference on Artificial Intelligence (IJCAI-19)*, pages 2592–2599, Macau, China, 2019. See arXiv:1905.12767 for a related and expanded paper (with additional material and authors).

[51] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. Drn: A deep reinforcement learning framework for news recom-

mendation. In *Proceedings of the 2018 world wide web conference*, pages 167–176, 2018.

[52] Bichen Shi, Makbule Gulcin Ozsoy, Neil Hurley, Barry Smyth, Elias Z. Tragos, James Geraci, and Aonghus Lawlor. Pyrecgym: a reinforcement learning gym for recommender systems. In Toine Bogers, Alan Said, Peter Brusilovsky, and Domonkos Tikk, editors, *Proceedings of the 13th ACM Conference on Recommender Systems, RecSys 2019, Copenhagen, Denmark, September 16-20, 2019*, pages 491–495. ACM, 2019.

[53] Benjamin Kaiser, Akos Csiszar, and Alexander Verl. Generative models for direct generation of cnc toolpaths. In *2018 25th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*, pages 1–6, 2018.

[54] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[55] Marlesson R. O. Santana, Luckeciano C. Melo, Fernando H. F. Camargo, Bruno Brandão, Anderson Soares, Renan M. Oliveira, and Sandor Caetano. Mars-gym: A gym framework to model, train, and evaluate recommender systems for marketplaces. *CoRR*, abs/2010.07035, 2020.

[56] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, 2016.

[57] Jesse Vig, Shilad Sen, and John Riedl. The tag genome: Encoding community knowledge to support novel interaction. *ACM Trans. Interact. Intell. Syst.*, 2(3):13:1–13:44, 2012.

[58] Alexander D'Amour, Hansa Srinivasan, James Atwood, Pallavi Baljekar, D. Sculley, and Yoni Halpern. Fairness is not static: deeper understanding of long term fairness via simulation studies. In Mireille Hildebrandt, Carlos Castillo, L. Elisa Celis, Salvatore Ruggieri, Linnet Taylor, and Gabriela Zanfir-Fortuna, editors, *FAT* '20: Conference on Fairness, Accountability, and Transparency, Barcelona, Spain, January 27-30, 2020*, pages 525–534. ACM, 2020.

[59] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *J. Mach. Learn. Res.*, 22:268:1–268:8, 2021.

[60] Benjamin Recht, Christopher Ré, Stephen J. Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*, pages 693–701, 2011.

[61] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[62] François Chollet et al. Keras. `https://keras.io`, 2015.

[63] Raymond Yuan. Deep reinforcement learning: Playing cartpole through asynchronous advantage actor critic (a3c) with tf.keras and eager execution. `https://blog.tensorflow.org/2018/07/deep-reinforcement-learning-keras-eager-execution.html`, 2018.

[64] Farzaneh Rajabi and Jack Siyuan He. Click-through rate prediction using graph neural networks and online learning. *CoRR*, abs/2105.03811, 2021.

[65] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. A contextual-bandit approach to personalized news article recommendation. In Michael Rappa, Paul Jones, Juliana Freire, and Soumen Chakrabarti, editors, *Proceedings of the 19th International Conference on World Wide Web, WWW 2010, Raleigh, North Carolina, USA, April 26-30, 2010*, pages 661–670. ACM, 2010.

[66] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. DRN: A deep reinforcement learning framework for news recommendation. In Pierre-Antoine Champin, Fabien Gandon, Mounia Lalmas, and Panagiotis G. Ipeirotis, editors, *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, pages 167–176. ACM, 2018.

[67] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.

[68] Hua Zheng, Dong Wang, Qi Zhang, Hang Li, and Tinghao Yang. Do clicks measure recommendation relevancy?: an empirical user study. In Xavier Amatriain, Marc Torrens, Paul Resnick, and Markus Zanker, editors, *Proceedings of the 2010 ACM Conference on Recommender Systems, RecSys 2010, Barcelona, Spain, September 26-30, 2010*, pages 249–252. ACM, 2010.