

Final Master Thesis

**Double master's degree in Industrial engineering and
automatic and robotics**

**Deep Reinforcement Learning for Recommender
Systems**

MEMORY

Author: Héctor Izquierdo Enfedaque
Director: Cecilio Angulo Bahón
Date: 09/2021



Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona



Abstract

Recommender Systems aim to help customers find content of their interest by presenting them suggestions they are most likely to prefer. Reinforcement Learning, a Machine Learning paradigm where agents learn by interaction which actions to perform in an environment so as to maximize a reward, can be trained to give good recommendations.

One of the problems when working with Reinforcement Learning algorithms is the dimensionality explosion, especially in the observation space. On the other hand, Industrial recommender systems deal with extremely large observation spaces. New Deep Reinforcement Learning algorithms can deal with this problem, but they are mainly focused on images. A new technique has been developed able to convert raw data into images, enabling DRL algorithms to be properly applied. This project addresses this line of investigation.

The contributions of the project are: (1) defining a generalization of the Markov Decision Process formulation for Recommender Systems, (2) defining a way to express the observation as an image, and (3) demonstrating the use of both concepts by addressing a particular Recommender System case through Reinforcement Learning.

Results show how the trained agents offer better recommendations than the arbitrary choice. However, the system does not achieve a great performance mainly due to the lack of interactions in the dataset.

Summary

SUMMARY	3
1. GLOSSARY	5
2. INTRODUCTION	6
2.1. Scope and methodology.....	6
2.2. Objectives.....	7
3. PROJECT MANAGEMENT	8
3.1. Planification	8
3.2. Methodology.....	10
3.3. Tools.....	11
3.3.1. Git and GitHub	11
3.3.2. Visual Studio Code.....	11
3.3.3. Google Colaboratory	11
3.4. Economic analysis.....	11
3.4.1. Staff cost.....	12
3.4.2. Tools cost.....	12
3.4.3. Supplies cost.....	13
3.4.4. Final budget	13
3.5. Environmental analysis	13
4. REINFORCEMENT LEARNING	15
4.1. Key concepts.....	15
4.2. Model-free RL algorithms	17
4.3. RL problem as a finite Markov decision process.....	17
4.4. Examples.....	18
4.5. Deep reinforcement learning.....	18
5. RELATED WORK	20
5.1. Content-based filtering	20
5.2. Collaborative filtering	21
5.3. Deep neural networks	22
5.4. Reinforcement learning in recommender systems.....	24
5.4.1. Model-based approaches.....	25
5.4.2. Model-free approaches	26
5.4.3. Summary	28
6. PROBLEM FORMULATION	31

6.1. Recommender systems	31
6.2. From sequential decision-making problems to MDPs	31
6.3. MDP generalization.....	32
7. BUILDING A RECOMMENDER SYSTEM _____	34
7.1. Case study	34
7.1.1. Dataset selection	34
7.1.2. Problem definition	36
7.2. MDP formulation	40
7.2.1. S definition	40
7.2.2. A definition	41
7.2.3. R definition.....	41
7.3. RS Implementation	42
7.3.1. Data preprocessing.....	42
7.3.2. Environment implementation.....	45
7.3.3. Agent implementation	48
7.4. Results	52
7.4.1. Reinforcement Learning approach.....	53
7.4.2. Behavioral Cloning approach	56
8. CONCLUSIONS AND FURTHER WORK _____	60
BIBLIOGRAPHY _____	62

1. Glossary

RS: Recommender Systems

ML: Machine Learning

DNN: Deep Neural Network

DRL: Deep Reinforcement Learning

MDP: Markov Decision Process

RL: Reinforcement Learning

CNN: Convolutional Neural Network

GPU: Graphics Processing Unit

TPU: Tensor Processing Unit

A2C: Advantage Actor Critic

PPO: Proximal Policy Optimization

TRPO: Trust Region Policy Optimization

SB: Stable Baselines

BC: Behavioral Cloning

2. Introduction

Recommender systems (RS) aim to help customers find content of their interest by presenting them suggestions they are most likely to prefer. Indeed, humans tend to be less satisfied with the choices they make the more alternatives they have to choose from [4]. A RS can help reducing the number of options that are presented to the user. To do so, a Machine Learning (ML) based recommender model determines how similar videos and apps are to other things you like and then serves up a recommendation.

They are present in the day to day of many people. Amazon recommends similar items to the ones the user has already bought, Netflix proposes new movies and shows, LinkedIn suggests jobs, ..., all of them use recommender engines to help users discover new content.

Helping users find content not only improves customer satisfaction, but also revenue for those business who do it. According to Google, 40% of app installs on Google Play and 60% of watch time on YouTube come from recommenders [11]. In other sites, like Amazon, it reaches 35% of what consumers purchase [16], while on Netflix up to two out of three hours streamed are discovered through their recommender system. [5]

RS were firstly built based on either or both content-based filtering and collaborative filtering. Later, they started using Deep Neural Networks (DNN). More recently, Reinforcement Learning (RL) has also started to be used. In this project, we will go through RL based RS.

2.1. Scope and methodology

One of the problems when working with RL algorithms is the dimensionality explosion, especially in the observation space. New Deep Reinforcement Learning (DRL) algorithms can deal with this problem, but they are mainly focused on images. A new technique has been developed able to convert raw data into images, enabling DRL algorithms to be properly applied. Two projects have followed this line of research: Ferran Velasco used it to optimize industrial planning [9], while David Valero did it in a resource allocation problem [6]. This project will explore this line for a RS.

The project will focus on how to express the recommendation problem in order to solve it through the RL framework. This includes both defining the RS problem from the RL perspective, and implementing the necessary code to, given a RS, express it in a format that later allows to implement the RL algorithms.

First, RL will be introduced. This includes presenting RL inside the ML theory, explaining its

core ideas and formulating the problem as a Markov Decision Process (MDP). Also, how DNN can help dealing with dimensionality explosion will be addressed, including converting raw data into images.

Then, an analysis of the main proposed solutions to the recommendation problem will be carried on. This includes inspecting the latest RL-based RS proposals thoroughly, presenting how they approach the problem and how they formalize it.

After inspecting the state-of-the-art RL-based RS proposals and seeing how they formulate the problem, a generalization of the RS MDP formulation will be presented.

Finally, a particular case will be solved through RL.

2.2. Objectives

RL problems are usually formulated as an MDP. MDP can be described in many ways. In fact, RL-based RS related works propose different MDP formulations for the recommender problem. The first objectives of this project are to find the elements in common that these formulations have and to propose a generalization that encompasses them.

As it has been said, DRL algorithms can deal with dimensionality explosion, but they are mainly focused on images. To explode the advantages of these algorithms, data should be expressed as images. Hence, the second objective of this project is to express the problem state as an image, to later be able to use those algorithms.

Finally, the last objective is to put into practice all these theoretical concepts solving a particular case through RL theory. That is to say: find a particular case of RS, formulate it using the proposed MDP generalization, express its states as images and train an RL algorithm to give proper recommendations.

3. Project management

The objective of this chapter is to explain the aspects related to the management of the project, including how tasks are planned and managed, which methodologies and tools are used how much the development is expected to cost and which environmental impact has its development.

3.1. Planification

In order to achieve the goal and the objectives, the project is organized in several stages of development. Some stages, like the project management or the documentation, are developed throughout project. Others are dependent on each other and need the previous one to be completed before starting. Each stage includes several activities and has an assigned time of dedication.

The project will begin on February 15, 2021 and its completion is scheduled for September 16, 2021. The project is estimated to involve 720 hours of work, so the mean weekly dedication will be about 24 hours for a period of 30 weeks.

The following sections are dedicated to explain each of the stages, including the activities involved and the dedication time.

Project management

Project management activities are those related to the definition of goals and objectives, with tasks planning itself, with the project follow up and others not related to the subject of work itself. This stage is estimated in 24 hours and includes the following activities:

- PM 1. Scope, methodology and objectives definition.
- PM 2. Planning. Task dedicated to defining, planning and organizing project stages and tasks.
- PM 3. Economic analysis.
- PM 4. Follow up meetings.

State of the art study

The objective of this stage is to study the theoretical concepts required to implement a RL based RS. The estimated time for this stage is 144 hours and includes the following activities:

- SA 1. RL theory. Study the RL theory, including the MDP formulation.
- SA 2. RS state of the art. Study common RS approaches, including those based on

RL.

Problem formulation

Based on the previous acquired knowledge, the objective here is to formulate the RS problem in order to be solved by the RL theory. The activities involved are estimated to be finished in 24 hours and are the following:

- PF 1. Recommender systems problem formulation.
- PF 2. Recommender systems as MDP formulation.

Proof of concept

This stage consists of implementing a recommender system through reinforcement learning. This is the most time demanding task of the project, with a workload of 406 hours. The following activities are involved:

- PC 1. Recommender system selection. Select a recommender system dataset (if possible, a real case) in which to apply RL theory.
- PC 2. RS inside the MDP formulation. Define the case study inside the proposed MDP formulation. Implement the necessary code to, given the case study dataset, return the information as defined in an MDP and demanded by the RL framework.
- PC 3. Algorithm implementation. Study the possible RL algorithms, select the algorithm for the case study and implement the necessary code.
- PC 4. Learning process. Train the RL algorithm, including hyperparameters tuning.
- PC 5. Results analysis. Analyze the obtained results and extract conclusions.

Documentation

The last phase of the project refers to the documentation tasks, for which about 120 hours are estimated. The following activities are involved:

- DOC 1. Report writing. Write the project report.
- DOC 2. Defense preparation. Prepare the documentation for the defense of the project.

In order to represent the different scheduled stages and sets of tasks, a Gantt chart has been designed. A Gantt chart is a graphical tool whose objective is to expose the expected time of dedication for different tasks or activities over a given total time.

The project Gantt diagram can be seen in Figure 1.

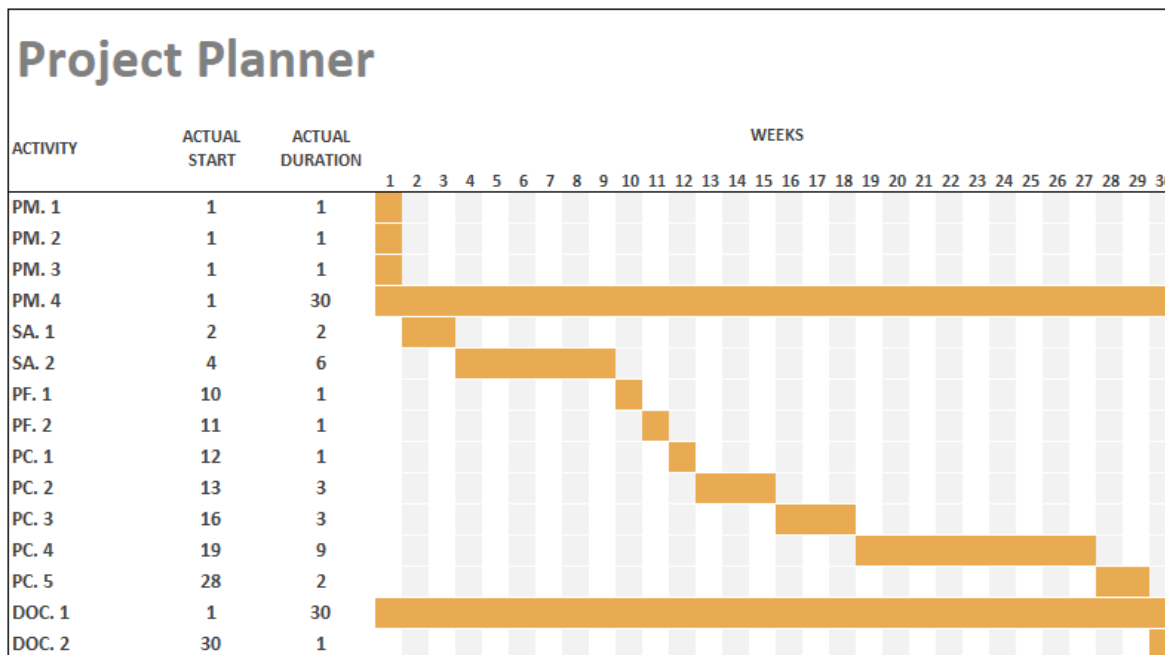


Figure 1. Project Gantt diagram

3.2. Methodology

Most of the activities previously described are too long to handle directly. Therefore, in the day to day work they will be divided into smaller tasks. In order to organize and prioritize them, the Kanban methodology will be used. In Kanban, tasks are visually placed in a board with *to do*, *in progress* and *done* columns. As a Kanban board, the GitHub project board will be used. As explained later, GitHub will also be used to host the code. A screenshot of the used Kanban board can be seen in Figure 2.

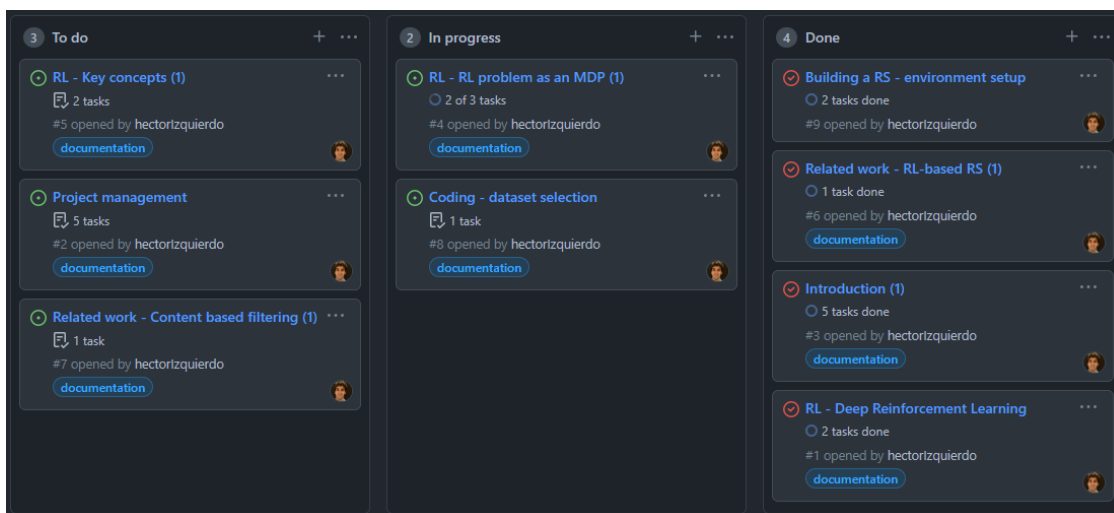


Figure 2. Project Kanban board

3.3. Tools

To make the project possible, in addition to the usual (Microsoft Office, ...), some specific tools have been used.

3.3.1. Git and GitHub

In order to track and manage the project code, a version control system will be used. Version control software keeps track of every modification to the code in a repository. If a mistake is made, one can turn back and compare earlier versions of the code to help fix the bug. The most widely used version control system is Git (<https://git-scm.com/>). As a code hosting platform, GitHub (<https://github.com/>) will be used.

3.3.2. Visual Studio Code

Visual Studio Code (<https://code.visualstudio.com/>) is a source code editor which runs on Desktop. It also works with Git, allowing staging files, committing, pushing or pulling from the GitHub repository.

3.3.3. Google Colaboratory

Google Colaboratory (or Colab) is a service that allows anybody to code and execute Python code through the browser. Colab notebooks are Jupyter notebooks hosted on Google's cloud servers, allowing to use the power of Google hardware, including Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs) [12].

GPUs are optimized for training artificial intelligence and deep learning models as they can process multiple computations simultaneously thanks to their highly parallel architecture. Therefore, they will allow to train the DRL algorithm so much faster than if using a CPU. Also, Colab can be integrated with Git.

3.4. Economic analysis

The objective of this section is to define the budget of the project. Costs can be divided into three categories:

- Staff. Research and development engineer (author) and director (college professor) salaries.
- Tools. Tools used by the staff, including equipment and software.
- Supplies. Supplies used by the equipment.

3.4.1. Staff cost

According to LinkedIn, a Research and Development Engineer salary is around 32.000 €/year [18]. Meanwhile, according to the UPC transparency portal, a Head of Investigation salary earns around 41.000€/year [35]. Taking into account 1750 hours of annual work, salaries are 18,29 €/h and 23,43 €/h, respectively. However, the cost for a Company is around 30% over the salary. Table 1 sums up the staff costs:

Role	Salary (€/h)	Cost (€/h)	Time (h)	Total (€)
R&D engineer	18,29	23,77	720	17.115,43
Head of investigation	23,43	30,46	10	304,57
Total				17.420,00

Table 1. Staff budget

3.4.2. Tools cost

The main tool that has been used to develop the project is a laptop. As for software, two of those that have been used are paid licenses: Microsoft Office and Google Colaboratory.

The laptop and the Microsoft Office package have not been used exclusively for the project. Therefore, only the time of use related to the project will be imputed. To do so, the following formula will be used:

$$cost = \frac{acquisition\ cost\ [€]}{useful\ life\ [years]} \times time\ attributable\ to\ the\ project\ [years]$$

As a *useful life*, 4 years will be considered. As time attributable to the project, 0,58 years (7 months). Considering acquisition costs of 1400 € and 150 € for the laptop and the Microsoft Office, the costs attributable to the project are 204,17 € and 21,88 €, respectively.

On the other hand, the Google Colaboratory subscription costs 8,76 €/month. In three months, the total amounts to 26,28€. Table 2 sums up the tools cost:

Tool	Cost (€)
Laptop	204,17
Microsoft Office	21,88
Google Colaboratory	26,28
Total	252,33

Table 2. Tools budget

3.4.3. Supplies cost

In the case of this project, only the laptop's electrical consumption can be directly attributable to the project. Table 3 sums up this cost:

Equipment	Consumption (KWh)	Time spent (h)	Cost KWh (€)	Cost (€)
Laptop	0,01	720	0,113	0,82

Table 3. Supplies budget

3.4.4. Final budget

Taking into account the previous defined costs, the total budget amounts to 17.673,15€ (see Table 4).

Concept	Cost (€)
Staff	17.420,00
Tools	252,33
Supplies	0,82
Total	17.673,15

Table 4. Final budget

3.5. Environmental analysis

The objective of this section is to analyze the environmental impact of the project, both that of the development itself and that of implementing the proposed solution.

The development impact can be estimated as the sum of two impacts: the impact of the hardware used (in terms of manufacturing) and the impact of the activities carried out with it. The first one is also the sum of the manufacturing impacts of the laptop and the peripheral. On the other hand, this equipment has not been used only for this project but for many other tasks. Therefore, the impact is proportional to the use dedicated to the project of these devices in relation to their useful life. The impact of the activities comes mainly from the electrical consumption of these devices.

Another relevant impact is the one caused by amount of computational power required to run RL training models on Google Colab. However, doing this training in the Cloud is more efficient in fact than running it on the local host. The cloud supports the operation of many products

simultaneously, allowing it to be more efficient in distributing resources among different users. Also, all the energy used by Google comes from renewable energy.

It is difficult on the other hand to estimate the impact of the implementation at a high scale. Training RL agents from scratch is very inefficient at first, since the algorithm should find a solution of the problem with a priori no knowledge of the environment. However, if the proposed RL agent ends up giving better recommendations than actual systems, one could say people will spend less time looking for a recommendation and that would in fact save energy (taking into account that, for instance, people are not going to watch more movies because of the good recommendations given than if worse recommendations were done; otherwise the impact would increase).

4. Reinforcement learning

Reinforcement learning (RL) is a machine learning (ML) paradigm where agents learn which actions to perform in an environment so as to maximize a reward. Agents are not told which actions to take, but instead must discover which of them yield the highest reward by trial-and-error. In most cases, actions may affect not only the immediate reward but also the next situations and, through that, all subsequent rewards. Trial-and-error search and delayed reward are the two most important distinguishing features of reinforcement learning.

Reinforcement learning is different from supervised learning, which learns from a training set of labeled examples and tries to extrapolate its responses so that it acts correctly in other situations. Although this approach may be useful when facing some situations (e.g., image classification), in interactive problems it is often impractical to obtain labelled examples representative of all the situations the agent has to deal with. It is also slightly different from unsupervised learning, which usually involves finding hidden structures in collections of unlabeled data. Despite finding those structures might be helpful in RL, it does not address the problem of maximizing a reward signal.

4.1. Key concepts

The agent and the environment continuously interact with each other. At each time step t , the agent receives an observation s_t^1 from the environment and takes an action a_t on it based on its policy $\pi(a_t/s_t)$. One time step later, the agent receives a numerical reward r_{t+1} and a new observation s_{t+1} from the environment. The goal of the agent is to improve the policy so as to maximize the cumulative reward, called return.

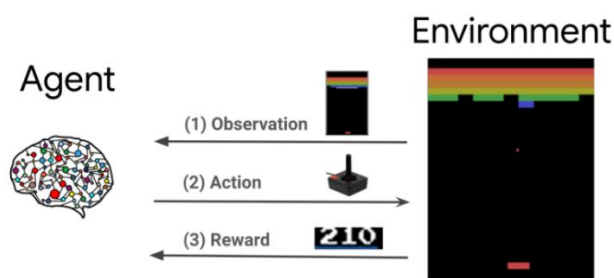


Figure 3. Agent-environment interaction. Source: [34]

¹ In most literature, state and observation are used interchangeably and observation is denoted as s . However, it is important to distinguish between the state of the environment and the observation, which is the part of the environment state that the agent can see. The action and the policy are not conditioned to the state but to the observation of the state the agent has access to.

The key elements of the RL problem are the environment, which represents the problem to be solved, and the agent, which represents the learning algorithm. Beyond them, one can identify four other important elements: the policy, the reward signal, the value function and, optionally, the model.

The policy defines the agent's way of behaving at a given time. It represents a mapping from perceived states of the environment (observations) to actions to be taken when in those states. Policies can either be deterministic or stochastic.

The reward signal defines the goal of the RL problem. On each time step, the environment sends to the agent a single reward indicating whether the action taken was good or bad for him. If the received reward is low, then the policy may be changed to select some other action in that situation in the future. In general, reward signals depend on the state of the environment and the action taken. The sum of rewards over time is called return, and is what the agent seeks to maximize. The return (Eq. 1) is the total accumulated return from time step t , where $\gamma \in [0,1]$ is a discount factor that benefits immediate rewards with respect to the future ones.

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \quad \text{Eq. 1}$$

The value function (Eq. 2) defines the total amount of reward an agent can expect to accumulate over the future, starting from a specific state. Whereas the reward signal indicates how good the previous action taken was, the value function indicates how good the current state is, after taking into account the states and rewards are likely to come from that point on. Similarly, the action value function (Eq. 3) is the expected return when choosing action a in state s .

$$V(s) = \mathbb{E} [R_t | s_t = s] \quad \text{Eq. 2}$$

$$Q(s, a) = \mathbb{E} [R_t | s_t = s, a] \quad \text{Eq. 3}$$

Another element present in some RL systems is a model of the environment. The model gives response to questions about how the environment will behave. For instance, it might predict the next state and reward given the current state and a performed action. Methods for solving RL problems that use models are called model-based methods, whereas methods for solving those that do not are called model-free. To be useful, models have to be reasonably accurate. That's why model-free methods can have advantages over model-based methods when the difficulty of the problem is to construct a sufficiently accurate model.

4.2. Model-free RL algorithms

Model-free RL problems are usually approached in two ways: policy-based and value-based methods (also called Q-learning methods).

In value-based methods, the action value function is approximated by $Q(s, a; \theta)$, typically using an objective function based on the Bellman equation to optimize the parameters. This optimization is usually performed off-policy, which means the data used in the update was collected at any point during the training, regardless of which policy was followed at that time. The resulting policy is obtained by selecting the actions that yield the highest expected reward when being in a state s (Eq. 4).

$$a(s) = \arg \max_a Q(s, a; \theta) \quad \text{Eq. 4}$$

On the other hand, policy-based methods directly parameterize the policy as $\pi(a|s; \theta)$ and update the parameters θ either by gradient ascent directly on the objective function (typically the expected reward), or by maximizing local approximations on the objective function also. This update is usually performed on-policy, which means the data used in the update was collected following the most recent policy.

Policy-based methods directly optimize the policy, and they are usually more stable and reliable. Meanwhile, value-based methods indirectly optimize the agent by training the approximator to satisfy a separate objective function, and although not being stable, they are more sample efficient, since they can reuse data more effectively.

4.3. RL problem as a finite Markov decision process

The problem of reinforcement learning is often presented as a finite MDP. MDP are a formalization of sequential decision-making problems, where actions influence not just the immediate states and rewards, but also the subsequent ones.

An MDP is a 5-tuple, $\langle S, A, R, P, \gamma \rangle$, where:

- S is the set of valid states
- A is the set of valid actions
- $R: S \times A \rightarrow R$ is the reward function, with $r(s, a)$ being the reward received when applying action a in state s .
- $P: S \times R \times S \times A \rightarrow P(S)$ is the transition probability function, with $p(s', r|s, a)$ being the probability of transitioning from state s into state s' when applying action a , giving also a reward r .

- γ is the discounting factor for future rewards

In a Markov decision process, the probabilities given by p completely characterize the environment's dynamics. That is, the probability of each possible value for the state and the reward depends only on the immediately preceding state and action and not on the earlier ones. The state must include then information about all aspects of the past that are necessary for the future. If it does, then the state is said to have the Markov property.

When an MDP has finite state, action and reward sets, it constitutes a finite MDP. Much of the RL theory is restricted to finite MDP.

4.4. Examples

RL algorithms were first limited to low-dimensional problems due to limitations in memory and computation capacity. They succeeded for instance in [10], where a neural network learned to play backgammon by playing against itself. One of the first success histories using RL mixed with deep neural networks was the development of an algorithm that could play a wide range of Atari 2600 video games directly from raw pixels [21][22]. Later on, it has also succeeded at sophisticated strategy games such as chess [31], Go [31][32] or Dota.

They have also been applied in robotics, where control policies for robots can be learned directly from camera and sensor inputs. [1]

4.5. Deep reinforcement learning

Reinforcement learning algorithms often need to deal with high dimensional state and action spaces. This is known as the curse of dimensionality. Deep neural networks (DNN) can help overcoming these problems, since they can learn low-dimensional feature representations from high dimensional data and have strong function approximation properties. The application of DNN to RL is known as Deep Reinforcement Learning (DRL).

One of the DNN architectures is the convolutional neural network (CNN). A CNN is an artificial neural network that uses convolutional layers to filter inputs for useful information. These layers combine a feature map (the input data) with a kernel (a filter) to obtain a transformed input map. It can be seen as a filter operation, where the kernel filters the input to obtain certain information.

CNNs are composed of an input, an output, and one or more hidden layers. Hidden layers are a combination of convolution, pooling and normalization layers, among others. CNN are characterized by having the neurons of their layers arranged in three dimensions, allowing the

network to extract useful information from input data arranged accordingly. [24]

Traditional ML techniques usually considered the input as a vector with independent features, where the order of features had no direct impact in the classification problem. As opposed, CNN can capture additional information by considering the positioning of neighboring pixels, uncovering hidden patterns and helping understand relationships between features. The second advantage is that hidden layers allow feature extraction and classification without additional techniques, directly from the raw elements. Third, CNN architectures find higher order statistics and nonlinear correlations. Another characteristic is that each convolution neuron processes data in its limited subarea, allowing a deeper network while keeping the number of neurons. Also, many of them share weights, which turns into a reduction in the memory use. The use of GPUs (Graphics Processing Units) also allows complex models to be trained in a much faster and affordable way.

A lot of data is in a non-image form (a width x height x depth arrangement) and cannot be processed by CNNs. However, if non-image data is transformed into a well-organized image form, CNNs can be effectively used. This way of proceeding is explored in [17], where the researchers transform RNA sequences into 2D images to detect cancer in early stages by using CNNs. It is also used by [7] and [2], where the authors turn patient data also into 2D images to later generate fake patients through Generative Adversarial Networks (GANs). [30]

5. Related work

RS have been typically approached through content-based filtering, collaborative filtering or a combination of both. In the last years, these methods have started being substituted by DNN, since they can address most of their limitations. Recently, some papers introduced RL into this problem. The objective of this section is to explore the different methods RS are approached by, giving emphasis to the RL based ones.

5.1. Content-based filtering

Content-based filtering tries to predict about the interests of a user based on similar items the user likes. I.e., if user A watches a *Star Wars* movie, then the system can recommend other *Star Wars* movies to that user. To do so, the model relies on the user previous actions (the previous film watched) or explicit feedback (a rated film). Past interactions are usually stored in a feature matrix (see Figure 4), where each row represents an item and each column a feature the items may or may not have (in the case of binary features). The user is also represented in this matrix. Most of the times, items and users' features are embedded in a lower-dimensional space vector (see Figure 5), where some latent structure can be captured.





	Feature 1	Feature 2	Feature 3	Feature 4
	✓			✓
		✓	✓	
		✓		
	✓	✓		✓

Figure 4. Feature matrix example

In order to provide relevant recommendations, the model uses a similarity measure to rate each candidate item against the user and then decides which item to recommend based on the scores among the candidates. A similarity measure is a function that takes two feature vectors and returns a scalar value indicating how similar these vectors are. Common similarity functions are the cosine, the dot product or the Euclidean distance.

This kind of filtering can better capture the specific interests of a user and therefore recommend items that few users are interested in. Another advantage is that it does not need information about other users. In contrast, the feature representation requires some hand-engineering, which can limit how good the model is. Also, since it only gives recommendations based on past interactions, it is difficult for the model to recommend items beyond them.

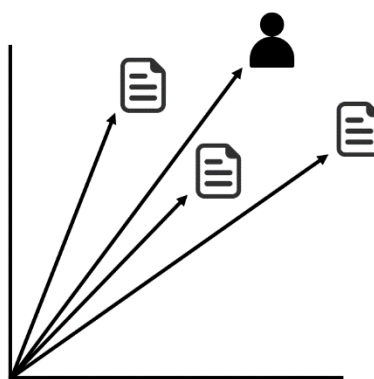
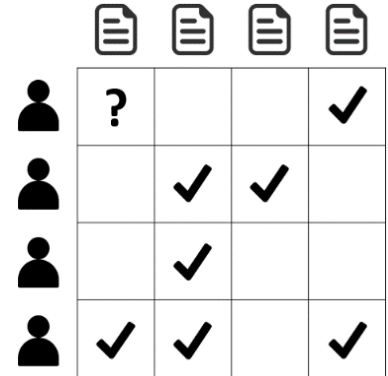


Figure 5. Previous feature matrix user and items represented in a lower dimensional space (2D)

5.2. Collaborative filtering

Collaborative filtering can address some of the content-based filtering limitations by using similarities between users and items simultaneously. That is, it tries to predict about the interests of a user based on the interests of many other users. That is, if user A is similar to user B, and user B likes *Star Wars* movies, then the system can recommend *Star Wars* movies to user A. Like content-based filtering, this method uses past interactions between users and items in order to produce new recommendations. These interactions are usually stored in a user-item feedback matrix (see Figure 6), where each row represents a user and each column an item.



The figure shows a 4x4 grid representing a user-item feedback matrix. Above the grid are four document icons representing items. To the left of the grid are four person icons representing users. The grid cells contain either a question mark, a checkmark, or are empty.

	?			✓
		✓	✓	
		✓		
	✓	✓		✓

Figure 6. User-item feedback matrix example.

Collaborative filtering is usually approached through matrix factorization. This method tries to find an underlying model that explains the interactions in order to make new predictions. To do so, it assumes there exists a low dimensional space of features in which users and items can be represented such that the interaction can be obtained by computing the dot product of both feature matrices.

Formally, given the feedback matrix A in $R^{m \times n}$, where m is the number of users and n is the number of items, the model learns:

- A user embedding matrix U in $R^{n \times l}$, where row l is the embedding for user u .
- An item embedding matrix V in $R^{m \times l}$, where column m is the embedding for item v .

The embeddings are learned such that the product UV^T is a good approximation of the feedback matrix A (see Figure 7). The problem can be solved by minimizing the mean square error (MSE), the squared Frobenius distance or through Singular Value Decomposition (SVD), among other methods.

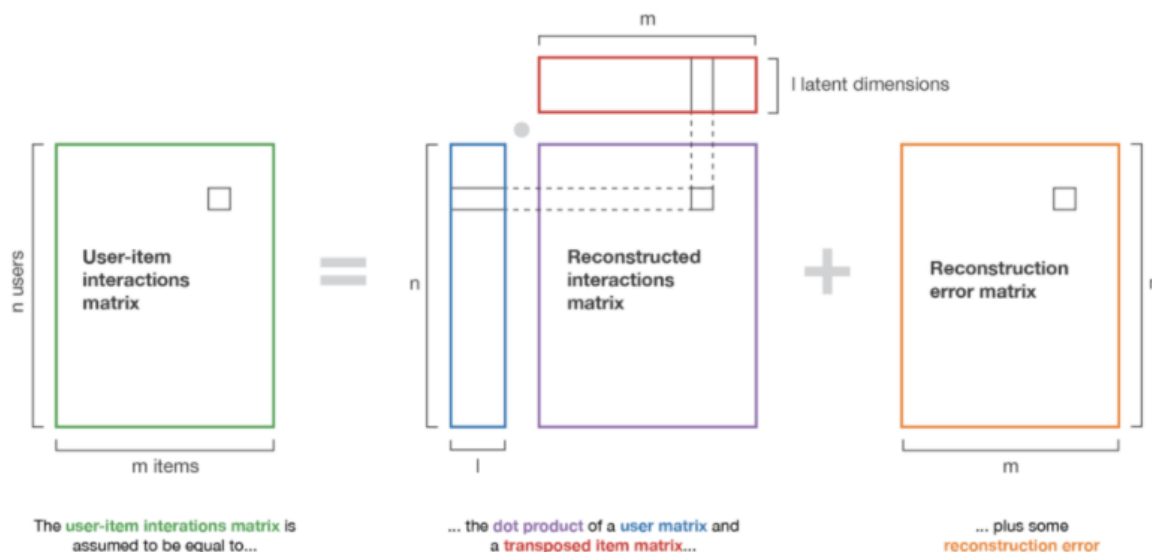


Figure 7. Matrix factorization optimization problem. Source: [3]

However, in real applications the feedback matrix A can be very sparse, which makes these methods perform poorly. In contrast, other methods such as Weighted Matrix Factorization (WMF) decomposes the objective function into a weighted sum of two sums: one over the observed items and another one over the unobserved ones. Common algorithms used to minimize the objective function include Stochastic Gradient Descent (SGD) and Weighted Alternating Last Squares (WALS), an iterative method specialized for this particular objective.

However, collaborative filtering has two main drawbacks:

- The prediction for a given user-item pair is calculated through the dot product of the corresponding embeddings. If new users or items are added after training the model, the system can't create an embedding for them. This issue can be addressed with heuristic methods.
- Sometimes other information about users or items is available. These are called side features, and including them improves the model. A generalization of WALS allows to include these features by building a block matrix from the feedback matrix A together with the side features matrixes and solving it.

5.3. Deep neural networks

DNN models can address matrix factorization limitations. They suffer far less from the cold-start problem, as they can easily give relevant suggestions to new users. They can also include side features through their layers, which can help capture specific interests and improve the relevance of the recommenders. [11]

One common approach is treating the problem as a multiclass prediction problem, where the input is the user query and the output is a vector whose values indicate the probability to interact with each item. A set of hidden layers map the input vector, denoted as x , to a more compact vector denoted by $\psi(x)$. Then, a softmax layer maps this vector to a probability distribution (see Figure 8). In order to train the model, a loss function compares the probability distribution with the ground truth.

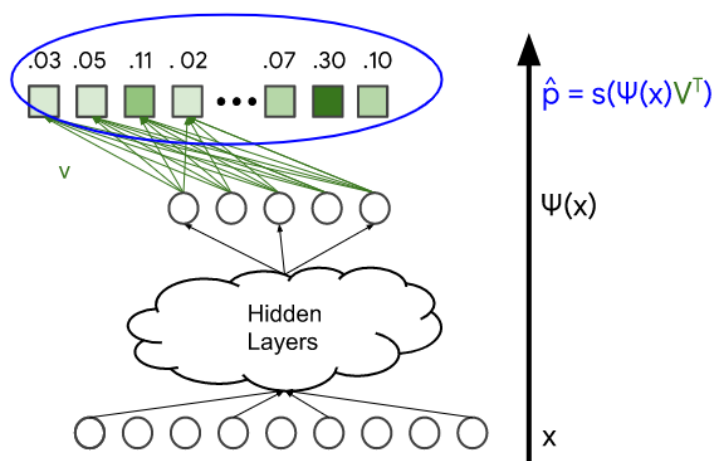


Figure 8. DNN schema. Input features are represented as x , while probability distribution is represented as p . Source: [11]

Instead of learning one embedding per user, as in the matrix factorization model, the system learns a mapping from the user feature vector to an embedding.

Another approach when using DNN in recommender systems consists on applying the same idea to the item side. That is, instead of learning one embedding per item, a neural network learns a nonlinear function that maps item features to an embedding. To do so, two neural networks are used:

- One neural network maps user features x_{user} to user embedding $\psi(x_{user}) \in R^d$
- One neural network maps item features x_{item} to item embedding $\phi(x_{item}) \in R^d$

User and item embeddings are then used as inputs in a classification model that gives the interaction value for the user-item pair (see Figure 9).

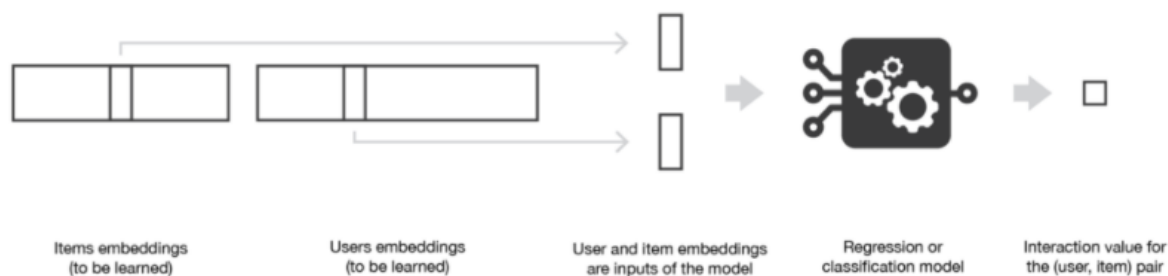


Figure 9. Interaction user-item through user and item embedding into a classification model. Source: [3]

YouTube uses DNN for video recommenders. Their system is comprised of two neural networks: one for candidate generation and another one for ranking. The candidate generation network uses user's activity history to retrieve a small subset of videos from the initial set. To do so, the problem is treated as a multiclass prediction problem. The task of the DNN is to learn the user embeddings as a function of the user's history and context that are useful for discriminating among videos with a SoftMax classifier. At serving time, the problem is reduced to a nearest neighbor search in the dot product space.

In a second stage, candidate videos are sorted to present a few best recommenders list to the user. The ranking network accomplishes this task by assigning a score to each video according to a desired objective function using a rich set of features describing the video and user. [26]

5.4. Reinforcement learning in recommender systems

Previous approaches adopted a static view of the recommendation process, predicting a user's immediate response to a recommendation without considering the sequential nature of the process and the long-term impact on the user future behavior.

Not considering the sequential nature of the process can be limiting, since user choices might be sequential by nature. E.g., booking an apartment near the beach after seeing others that were already reserved. Moreover, optimal recommendations may depend not only on user's past responses, but also on the order in which those responses were given. I.e., watching the sixth movie of the Star Wars saga after the fifth. [14]

A good recommender system should not only bring good recommendations, but also keep users interacting with the system, maximizing the long-term benefit. Thus, modeling a system taking into account current recommendations impact on the future allows to trade off user satisfaction in the near-term for longer-term benefit. RL methods are trained estimating the present value with delayed rewards and their objective is to maximize the long-term cumulative rewards. Therefore, they identify suggestions with low immediate reward that can lead to greater rewards through future recommendations. [19]

Also, one of the aspects to be designed in RL problems is the reward function. Since it can be freely designed, it is possible to define models that can serve multiple objectives at a time, such as novelty or diversity. [40]

As in general RL problems, there are mainly two kinds of RL methods for recommender: model-free and model-based. Model-free RL algorithms usually need to interact many times with the environment until the policy is trained enough to start providing the agent acceptable solutions. Doing those first interactions online is unfeasible in commercial systems, since recommenders with an under-trained policy would affect the user experience. A typical solution is learning off-policy from the logged implicit feedback. On the other hand, model-based RL methods aim to build a model to simulate the environment so that the agent can learn by interactions with it. These methods depend on the constructed simulator.

This section aims to take a look at the different proposals for RS using RL, focusing not only on the proposed algorithms, but on how the problem is posed from the RL point of view.

5.4.1. Model-based approaches

To better simulate the environment and the reward, Chen et al proposed a GAN formulation. The Generative Adversarial User Model estimates the user behavior dynamics and the reward function simultaneously through a unified min-max optimization algorithm. Using this model as the simulation environment, a cascading DQN algorithm allows to obtain the recommender policy.

When modeling the problem as an MDP, they proposed the state to be an ordered sequence of a user's historical selected items represented by an embedding of their feature vectors. The action of the recommender will correspond to a subset of k items chosen from the available items at that time. Finally, the reward function will be based on the user's long-term satisfaction after selecting the item when the subset was presented. [41]

Bai et al proposed a similar approach in [42]. Their method, named Interactive Recommender GAN (IRecGAN), models user-agent interaction for offline policy learning through a generative adversarial network. Then, they use REINFORCE for the agent's policy learning.

To represent the state when modeling the problem, clicked items are first projected into an embedding vector. The state is represented as an embedding of the click history, where the last embedding function can be GRU and LSTM. Each action gives a recommendation list of k items. In the example provided, they considered the reward to be 1 if the user stayed in the recommender system and 0 otherwise.

5.4.2. Model-free approaches

Zou et al. introduced a RL framework called FeedRec to directly optimize the user engagement (both instant and long-term) in feed streaming recommender. FeedRec includes two components: 1) a Q-Network with hierarchical LSTM (Long short-term memory) architecture takes charge of modeling complex user behaviors, and 2) a S-Network, which simulates the environment, assists the Q-Network and voids the instability of convergence in off-policy learning.

When formulating the problem, they considered the interaction process between the user and the environment as the sequence $x_t = \{u, (i_1, f_1, d_1), \dots, (i_t, f_t, d_t)\}$, where u is the user information, i_t is the item from the recommendable item set, f_t is the feedback provided by the user to the item i_t and d_t is the dwell time on the recommender. The state at time step t is designed as the browsing sequence $s_t = x_t - 1$. The initial state only contains the user information. At each time step, s_t is updated with the previous state and the tuple of recommended item, feedback and dwell time. Action a_t is the recommender of item i_t . The action space is updated by removing recommended items. [19]

Xin et al. propose a self-supervised reinforcement learning method for sequential recommender tasks. Their approach augments recommender models with two output layers: one for self-supervised learning and the other for RL. While the self-supervised layer with cross-entropy loss provides strong gradient signals for parameter updates, the RL layer which aims to introduce reward driven properties to the recommender. Based on such approach, they propose two frameworks namely Self-Supervised Q-learning (SQN) and Self-Supervised Actor-Critic (SAC).

When formulating the problem, they considered the user-item interaction sequence as $x_{1:t} = \{x_1, \dots, x_i, \dots, x_t - 1, x_t\}$, where x_i is the index of the interacted item at timestamp i . The state of a user at timestamp t can be represented as $s_t = G(x_{1:t})$ in S , where G is a generative model that maps the input sequence into a hidden state. Also, they consider the action a_t as recommending an item i at time t . They propose a flexible reward scheme depending on the scenario; e.g., for video recommender, a reward based according to the watching time. [40]

le et al. developed SlateQ, a novel decomposition technique for slate-based RL that allows for effective TD and Q-learning by estimating the long-term value (LTV) of a slate of items by directly using the estimated LTV of the individual items in the slate.

When formulating the problem, they considered the state s_t as the combination of user features (demographics, interests) and relevant user history or past behavior (items consumed, degree of engagement). As it has been said, they provide slates of items. Thus, an action a recommends a possible slate with k items. As a reward, they measure the expected degree

of user engagement within the items in the slate. [8]

Xiangyu Zhao et al. identified the importance of negative feedback (such as skipping an item) from the recommender procedure and provided an approach to capture it. They later proposed a deep reinforcement learning based framework (DEERS) that learns the optimal strategies by combining the negative feedback with the positive and tested it in an e-commerce scenario.

When modeling the problem as an MDP, they considered the state s_t as the recommender-user interaction at time t , in chronological order. Instead of using product's attributes, they treated each item as a word and the clicked items in one recommender session as a sentence. Then, they embedded those words into a low-dimensional vector. As an action, they considered recommending one item to the user at time t . The recommender agent receives immediate reward according to the user's feedback (click, skip or order the item). [38]

In another paper, Xiangyu Zhao et al. introduced a principled approach to generate a set of complementary items and properly display them in a recommender page simultaneously. They proposed a framework for page-wise recommender, DeepPage, to address this approach.

Their MDP formulation of the problem is very similar to one they presented in [38]. They proposed the same state space S and the same reward approach. As for the action space, they considered recommending a list of items at each time t , where K is the number of items. [39]

In [37], Xiangyu Zhao et al. proposed a similar approach that aimed to capture the relationship among recommended items and generate a list of complementary items to enhance performance. They propose a framework for list-wide recommenders based on DRL, LIRD, which can be applied in scenarios with large and dynamic item space. They also introduced an online user-agent interacting environment simulator, which can pre-train and evaluate model parameters offline before applying the model online. Their MDP formulation of the problem is the same as the one they presented in [38].

Chen et al. scaled a REINFORCE policy-gradient-based approach to learn the policy in an extremely large action space. They address biases when learning from logged data through incorporating a learned logging policy and a novel top- K off-policy correction. They conducted live experiments on YouTube.

Their MDP formulation of the problem considered the state as the user historical interactions with the system, recording the actions taken by the recommender as well as user feedback (such as clicks or watch time). The action at consists on recommending a set of videos and the reward is based on the immediate user response to the received set. [20]

Zheng et al. proposed a RL framework that applies a DQN structure and can take care of both

immediate and future reward. To help improving recommender accuracy, they considered user activeness as another form of user feedback. They propose also to apply a Dueling Bandit Gradient Descent in order to add some exploration while avoiding totally unrelated items. They deployed their algorithm in a commercial news recommender application.

Their MDP formulation considered the state as a combination of context features and user features. Context features describe the context when the news request happens (time, weekday, ...), while user features mainly describe the features of the news clicked by the user some timestamps before. The action a_t consists on recommending a list of k news. Finally, the reward takes into account immediate feedback (such as clicks) as well as future response (user activeness). [13]

5.4.3. Summary

Table 5 summarizes both the model-based and the model-free approaches seen so far:

Author	State	Action	Reward	Algorithm	Scenario
Chen et al. [41]	<i>An embedding of the historical sequence (or a truncated M-step sequence) of items clicked by the user represented by their feature vectors.</i>	<i>Recommending a subset of k items</i>	<i>Based on user utility. Unknown schema.</i>	<i>Deep Q Networks</i>	<i>Movie, music, business, e-commerce, news</i>
Bai et al. [42]	<i>Clicked items are projected into an embedding vector. The state is represented as an embedding of the click history</i>	<i>Recommending a subset of k items</i>	<i>In the example they provide, an immediate reward of 1 is given if the session continues</i>	<i>REINFORCE</i>	<i>E-commerce</i>
Zou et al. [19]	<i>User's information + user-item interaction (recommended item, feedback and dwell time) updated every timestep</i>	<i>Recommending an item</i>	<i>A combination of instant metrics (clicks) and delayed metrics (scans and return time)</i>	<i>Deep Q Networks</i>	<i>E-commerce</i>
Xin et al. [40]	<i>An embedding of user-item interaction history</i>	<i>Recommending an item</i>	<i>Flexible reward scheme</i>	<i>Deep Q Networks; a variation of SAC</i>	<i>E-commerce</i>
le et al. [8]	<i>A combination of user features and past behavior (past recommenders, items consumed, degree of engagement)</i>	<i>Recommending a subset of k items</i>	<i>Expected degree of engagement</i>	<i>TD</i>	<i>YouTube</i>

Xiangyu Zhao et al. [38]	<i>Recommenders-user interaction (in chronological order) embedded in a low dimensional vector</i>	<i>Recommending an item</i>	<i>Immediate reward according to the user's feedback (skip, click or order)</i>	<i>An approach based on Deep Q Networks</i>	<i>E-commerce</i>
Xiangyu Zhao et al. [39]	<i>Recommenders-user interaction (in chronological order) embedded in a low dimensional vector</i>	<i>Recommending a subset of k items</i>	<i>Immediate reward according to the user's feedback (skip, click or order)</i>	<i>Deep Q Networks</i>	<i>E-commerce</i>
Xiangyu Zhao et al. [37]	<i>Recommenders-user interaction (in chronological order) embedded in a low dimensional vector</i>	<i>Recommending a subset of k items</i>	<i>Immediate reward according to the user's feedback (skip, click or order)</i>	<i>Actor-Critic</i>	<i>E-commerce</i>
Chen et al. [20]	<i>User's historical interactions with the system</i>	<i>Recommending a subset of k items</i>	<i>Based on user's response (clicks, watch time)</i>	<i>REINFORCE</i>	<i>YouTube</i>
Zheng et al. [13]	<i>Context features (demographics, ...) + user features (interactions in the last k periods)</i>	<i>Recommending a subset of k items</i>	<i>Immediate response + long-term response (user activeness)</i>	<i>Deep Q Networks</i>	<i>News</i>

Table 5. RL approaches to RS summary

6. Problem formulation

In the previous chapter one could see how the RL-based RS proposals had a common way of formulating the problem: as an MDP. However, an MDP can be defined in many ways, and not all the authors did it the same. The objective of this chapter is to provide a generalization of the MDP formulation able to englobe as many as the RL-based RS current state of the art proposals.

First, the RS will be explained from a theoretical point of view. After that, the RS problem will be posed as a sequential decision-making problem, a necessary step to formulate it as an MDP. Finally, a generalization of the MDP formulation will be proposed, as well as how the previous state of the art articles can fit in this formulation.

6.1. Recommender systems

Recommender systems (RS) aim to help customers find content of their interest by presenting them suggestions they are most likely to prefer. To do so, an ML-based recommender model determines how similar videos and apps are to other things you like and then serves up a recommender

Mathematically, the recommender problem can be stated as follows: there is a set of users U and a set of items to recommend I . Temporal horizon is represented by the set of periods $T = \{1, \dots, m\}$. At time stamp $t \in T$, each user $u \in U$ is defined by a set of information s_{u_t} ; e.g., personal features, preferences, past seen items (including the feedback given to them), and so on. Each item $i \in I$ is also defined by a set of information s_{i_t} . The recommender system has to, at a given time stamp $t \in T$, assign an item (or a badge of items) $i \in I$ to a user $u \in U$, in order for that assignation to be the preferred according to a previously defined criteria (i.e., profit, user engagement, etc.).

6.2. From sequential decision-making problems to MDPs

As stated in chapter 3, MDPs are a formalization of sequential decision-making problems. In this kind of problems, the utility of the agent's actions does not depend on single decisions, but rather on the whole sequence of the agent's decisions. Moreover, much of the RL theory applies when the problem can be described as a finite MDP (an MDP with finite state, action, and reward sets).

As stated before, classical recommender systems adopted a static view of the

recommendation process, treating it as a prediction problem. However, the process of generating recommendations can be better seen as a sequential decision-making problem. At a certain time, the RS decides which items suggests to the user, who chooses which of them to take. Once the user makes a choice, a new list of recommendations is presented to him, and so on. The system decides which recommendations to offer based on how users have responded to past recommendations and that's where the sequentiality of the process lies. Moreover, as stated before, the optimal recommendation may depend not only on past interactions, but in the order in which they were performed.

Thus, the recommendation problem can be formulated as an MDP. The RS can be considered as an autonomous agent who is performing actions (giving recommendations) based on the environment's state (user information, past recommendations, past response, item features, ...), while maximizing a reward signal (making good recommendations to the user). Moreover, considering a finite set of available items to recommend, a finite set of user information and a limited reward, the problem can be defined as a finite MDP.

6.3. MDP generalization

As stated before, An MDP is by definition a five-tuple $M = \langle S, A, P, R, \gamma \rangle$, where S is the set of states, A is the set of actions, P is the state-transition function, R is the reward function and $\gamma \in [0, 1]$ is the discount factor. Making a generalization of the MDP-formulation can be reduced to generalizing the sets of states and actions and the reward function. The following generalization can be made:

The state $s \in S$ can be generalized as the combination of user features (age, gender, demographic data, ...) and past user-item interactions (seen items, response given, ...). User features can be expressed as the initial set x_0 and past user-item interactions (of the last $t - m$ periods) as the sequence $x_{m:t} = \{x_m, \dots, x_{t-1}, x_t\}$ where $m \in [1, t]$ is a period and t is the actual time stamp. The set including both type of features can be defined as $x_{0:t} = \{x_0, x_m, \dots, x_{t-1}, x_t\}$. Finally, the state at time t can be represented as $s_t = g(x_{0:t})$ where g is an embedding function (it can be the identity function) that maps the input into a (usually) lower space. At each time step t , the set is updated with the old state and the new set of information. Previous RL-based RS articles [41], [40], [38], [39], [37], [42], [20] don't consider user features, so they would have x_0 empty. With respect to the length of the system-user interaction, [41] considers truncating the sequence to the last $t - m$ periods, where m is different from 1. Finally, [19], [8], [20], [13] don't embed the information, which can be considered as embedding with an identity function.

On the other hand, actions available often depend on the state s , denoting $A(s)$ as the available actions when being in state s . The action $a \in A(s)$ can be generalized as recommending one or more items i from the item set I , that is, $a = (i_1, \dots, i_n)$, where n is the number of items recommended. $A(s)$ can be updated by removing up to n recommended items from $A(s_{t-1})$. Previous RL-based RS articles [41], [42], [8], [39], [37], [20], [13] consider recommending more than one item, while [19], [40] and [38] only provide one recommendation.

Finally, the reward function must be generalized. This function, denoted as $r(s, a)$, provides the reward when taking action a at state s . Depending on the recommender system scenario, different reward functions can be used. E.g., for video recommender, reward can be set according to the watching time, whereas in the e-commerce scenario, rewards can be given to purchase interactions. As it can be seen, the reward has a strong dependence on the application. Taking a look at previous commented articles, we can distinct between three approaches: rewarding based on instant, delayed, or a combination of both metrics. While [38], [39], [37] and [42] consider only immediate reward, only [8] considers delayed feedback alone. On the other hand, [41], [19], [20] and [13] combine both instant and delayed responses. Generally speaking, we will define the reward as a combination of instant feedback and long-term feedback.

7. Building a recommender system

In the first chapters one could see what RL consisted of, how the MDP formulation was key in the RL frame and in which ways RS are usually approached. Moreover, in the last one a general MDP formulation for the RS was proposed. The goal of this section is to put all these theoretical concepts into practice.

The objective now is to build a RS by using the RL theory. Training from scratch a RL-based RS online is not feasible, as it would give poor recommendations to users for a long period of time. Instead, these systems are usually pretrained offline, from previous records, and then online. Learning from logged data is however subject to biases, since there's only feedback on recommendations selected from the previous RS.

The first part of this chapter will be dedicated to select and present the case study. After that, the selected problem will be expressed inside the proposed MDP generalization. Finally, the case will be solved through RL.

7.1. Case study

This section is devoted to select the case study and to explain what it consists of.

7.1.1. Dataset selection

In order to select a dataset, some criteria must be first established:

- High number of features. As explained in 2., one of the objectives is to use DRL algorithms, which are mainly focused on images. These algorithms usually require a minimum image size. Moreover, the main advantage of CNNs is that is capable of extracting relationships in complex images where it seems there are none. In order to meet the algorithm requirements and to extract the truly potential of them a rich dataset is required.
- Timestamp information. As explained in 4, one of the advantages of RL techniques is the possibility of capturing delayed rewards. Also, another one is that they can capture how the interests of the user vary along time. It is therefore essential to have the temporal information.
- Real case scenario. Some datasets are specially constructed for education. If possible, a real case scenario would be preferred.

Apart from the mentioned requirements, there are other aspects that should be taken into account:

- Number of interactions. RL algorithms usually need lots of interactions in order to learn, since they start from zero. Therefore, a high number of interactions is needed. However, too many interactions will take too much resources when training (RAM, processor, time) and might difficult the process of obtaining results. In addition, the number of features should be taken into account: it is not the same trying to learn 1 feature with 10 interactions than 10 features with 10 interactions; the more features present, the more interactions needed.
- Number of items. As with the number of interactions, a high number of items will require so many resources and therefore make the training process unapproachable. On the other side, RS are normally intended to recommend items from a large number of candidates. Again, this parameter has to be looked at together with the number of interactions and features.

It is difficult in fact to establish right values for these two aspects. However, there's the option of taking a small subsample of the dataset if the training gets to slow or the solutions don't converge.

Based on previous requirements and taken into account both the number of interactions and the action space dimension, nearly 15 datasets were selected as candidates. Finally, the following options were considered:

- Retail Rocket. This dataset consists of a list of events (click, add to cart or purchase, along with the user, the item and the timestamp), a list of items properties and a category tree, from a real-world ecommerce website. This dataset was used by [40] to build a RS. The objective is to predict the item purchased.
- Goodreads. This set consists of a list of user-book interactions (user, book, if is it read / reviewed and the rating, although timestamp can be asked), books metadata (including author information, genres, and so many other features), and reviews metadata (including all the text written by the users about the books) from the book rating Goodreads app. The objective is to predict book ratings.
- Trivago. This dataset consists of a list of sessions (including timestamp, user, action type, item, device, filters used, and some other), and a list of item features from the well-known platform Trivago. The objective is to predict the selected item. It was published in the RecSys Challenge 2019 (<https://recsys.trivago.cloud/>).

Table 6 shows the number of interactions, items and features each dataset presents:

Dataset	Field	Interactions	Items	Features
Retail Rocket	Ecommerce	2,76 M	400 K	~10
Goodreads	Books	229 M	2,36 M	~50

Trivago	Accommodations	19 M	927K	~10
---------	----------------	------	------	-----

Table 6. Candidate datasets comparison

One of the advantages of the Retail Rocket set is that it has been used in one of the papers studied. Also, it has a reasonable number of features and items. With respect to Goodreads, the main strength this set presents is the rich information it presents. As a counterpart, it is perhaps a too complex set to start with. Regarding the Trivago dataset, one of the advantages is that, since it was released for the RecSys challenge, it is well-suited for building the RS. As in the Retail Rocket one, it has a reasonable number of features and items. In fact, the relation interactions/number of items is higher, which is better. Taking into account all the insights, the Trivago dataset is finally selected.

7.1.2. Problem definition

Trivago (<https://www.trivago.es/>) is a metasearch engine that compares and displays the accommodation offers and prices provided by numerous online booking websites. If a user clicks on a specific offer, Trivago receives a commission from the booking website. In total, trivago websites include a staggering 2.5 million hotels and other types of accommodation in 190 countries.

The data provided for the challenge is a small subset that consists of 1.202.064 sessions from 730.803 users and metadata from 927.142 accommodations. Sessions contain information about different user actions: item search, filter usage, destination search, ... The action in which the user clicks on a specific offer and then is redirected to the booking website is the click-out. At the time of a click-out, the engine is displaying the user 25 accommodations along with their prices and ordered by preferences. The objective is to predict these 25 accommodations for a subset of sessions.

Sessions are divided into a training and a test set. The training set contains sessions up to a specified time (split date). The test set contains information about sessions after the split date but is missing the information about the accommodations that have been clicked in the last part of the sessions. The required output is a list of 25 recommendations for each click-out ordered by preferences for the specific user. The higher the actually clicked item appears on the list, the better the recommendation engine performs.

Figure 10 illustrates the problem setting and the separation of the data into training and test sets:

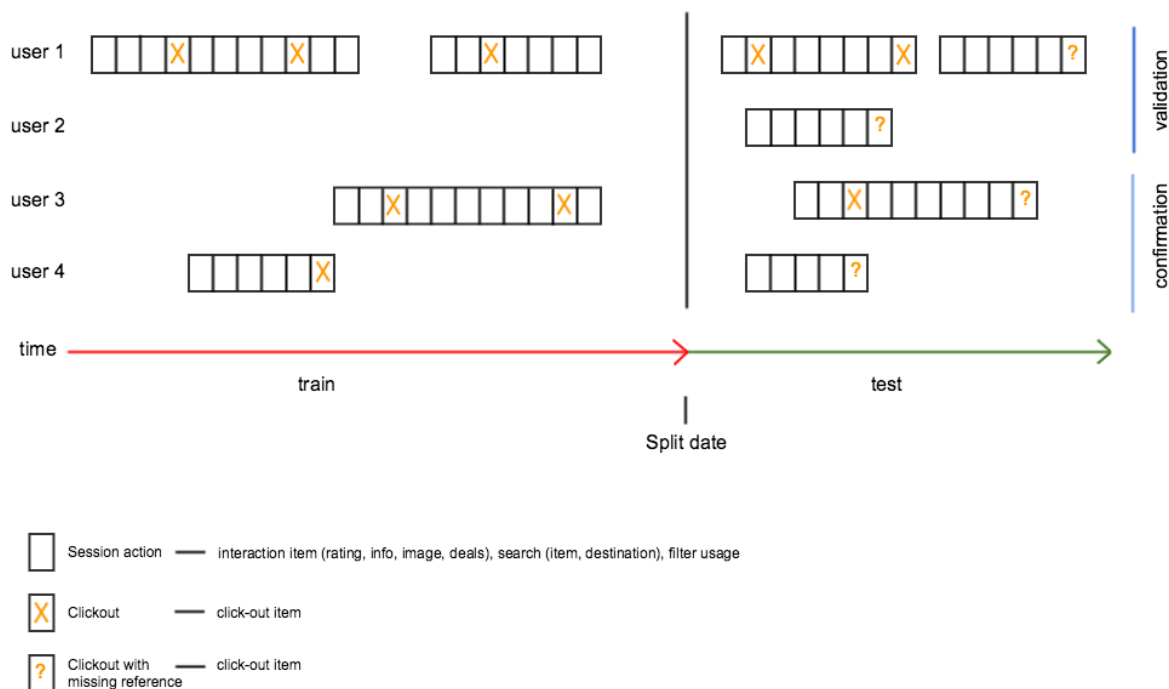


Figure 10. Problem setting schematic. Source: [28]

7.1.2.1. Sessions actions

Each session contains the following information:

- User ID: identifier of the user.
- Session ID: identifier of each session.
- Timestamp: UNIX timestamp for the time of the interaction.
- Step: step in the sequence of actions within the session.
- Action type: identifier of the action that has been taken by the user. The user can take 10 different actions:
 - click-out item: user makes a click-out on the item and gets forwarded to a partner website.
 - interaction item rating: user interacts with a rating or review of an item.
 - interaction item info: user interacts with item information.
 - interaction item image: user interacts with an image of an item.
 - interaction item deals: user clicks on the view more deals button.
 - change of sort order: user changes the sort order.
 - filter selection: user selects a filter.
 - search for item: user searches for an accommodation.
 - search for destination: user searches for a destination.
 - search for poi: user searches for a point of interest (POI).
- Reference: reference value of the action taken. All actions involving interactions with

items (click-out, rating, info, image, deals and search for) have the item ID as the reference value. Change of sort order has as reference the sort order description; filter selection the filter description; search for destination, the destination name; and search for POI, the POI name.

- Platform: country platform that was used for the search, e.g., trivago.es (SPAIN) or trivago.com (US).
- City: name of the current city of the search context.
- Device: device that was used for the search.
- Current filters: list of pipe-separated filters that were active at the given timestamp.
- Impressions: list of pipe-separated items that were displayed to the user at the time of a click-out.
- Prices: list of pipe-separated prices of the items that were displayed to the user at the time of a click-out.

A sample session is shown in Figure 11.

7.1.2.2. Item metadata

Item metadata includes information from every accommodation. Each of them includes the following information:

- Item ID: identifier of the accommodation as used in the reference values for item related action types.
- Properties: pipe-separated list of filters that are applicable for the given item.

A sample of the item metadata is shown in Table 7:

Item ID	Properties
32007	Hotel Country Hotel Pool Table Bathtub Satisfactory Rating Hiking Trail Luxury Hotel Terrace (Hotel) Shower Steam Room Cot Gym Ski Resort Good Rating Table Tennis Family Friendly Horse Riding Openable Windows Restaurant Solarium Sauna Bike Rental Car Park Central Heating

Table 7. Item metadata sample. Source: [28]

user_id	session_id	timestamp	step	action_type	reference	platform	city	device	current_filters	impressions	prices
93F7WGHBP03A	569f5ea70df51	1541543231	1	search for destination	Barcelona, Spain	US	Barcelona, Spain	desktop			
93F7WGHBP03A	569f5ea70df51	1541543269	2	filter selection	Focus on Distance	US	Barcelona, Spain	desktop	Focus on Distance		
93F7WGHBP03A	569f5ea70df51	1541543269	3	search for poi	Port de Barcelona	US	Barcelona, Spain	desktop	Focus on Distance		
93F7WGHBP03A	569f5ea70df51	1541543371	4	interaction item deals	40255	US	Barcelona, Spain	desktop			
93F7WGHBP03A	569f5ea70df51	1541543425	5	clickout item	40255	US	Barcelona, Spain	desktop		6744 40181 40630 84610 2282416 1258693 974937 147509 128238 7998246 40255 3058538 1637385 40285 147502 921707 40849 6757 12770 893733 685091 147522 40708 860451 6819	162 91 218 190 176 365 272 159 139 240 136 5099 164 116 90 192 191 213 109 178 131 128 168 101 331
93F7WGHBP03A	569f5ea70df51	1541543741	6	search for item	81770	US	Barcelona, Spain	desktop			
93F7WGHBP03A	569f5ea70df51	1541543770	7	interaction item info	81770	US	Barcelona, Spain	desktop			
93F7WGHBP03A	569f5ea70df51	1541543813	8	clickout item	81770	US	Barcelona, Spain	desktop		6832 40396 6621784 40197 6743 147488 40635 6177052 6742 1319782 40763 945255 83855 39937 1870125 1354432 6812 82400 40181 6834 81770 5056102 40797 923935 40284	347 245 199 65 359 233 227 270 294 625 208 174 121 217 226 616 293 166 91 198 274 272 123 130 131

Figure 11. Session sample. Source: [28]. In this session, a user from the US platform has used trivago on a desktop device. The actions in this session are the following:

- action type: search for destination, reference: Barcelona, Spain: User searches for Barcelona, Spain.
- action type: filter selection, reference: Focus on Distance: The 'focus on distance' filter is activated. At this point the current_filters column indicates that this is the only filter that is active.
- action type: search for poi, reference: Port de Barcelona: User searches for a point-of-interest (POI), the Port de Barcelona.
- action type: interaction item deals, reference: 40255: User viewed at the 'More Deals' button on item 40255. The 'focus on distance' filter is no longer activated.
- action type: click out item, reference: 40225: The user clicks out on item 40225. The full list of displayed items and their associated prices can be seen in the 'impressions' and 'price' columns.
- action type: search for item, reference: 81770: User searches for item 81770.
- action type: interaction item info, reference: 81770: User interacts with the item information of item 81770.
- action type: click out item, reference: 81770: User clicks out on item 81770. The full list of items and their associated prices can be seen in the 'impressions' and 'price' columns.

7.2. MDP formulation

Once the problem to be solved has been selected and before using the RL theory to construct a RS for it, it is convenient to express it as an MDP. As explained in 4.3, RL theory is applicable when the MDP formulation is possible. In addition, an MDP generalization for RS problems was previously done. The goal of this section is to fit the MDP expression in the aforementioned generalization.

As stated before, expressing a problem as an MDP means defining the five-tuple $M = \langle S, A, P, R, \gamma \rangle$, where S is the set of states, A is the set of actions, P is the state-transition function, R is the reward function and $\gamma \in [0, 1]$ is the discount factor. As will be explained later, the RL algorithm will be limited to learn from the dataset and not the ground truth, which brings certain limitations. One of them is that P is in fact already defined: the action determined by the algorithm will not affect the next state, which is already written on the next line in the dataset (anyway, defining P would mean having a model of the problem, turning it into a model-based instead of a model-free). On the other side, γ is usually set to 0,99. Therefore, the MDP definition is reduced to defining S , A and R .

7.2.1. S definition

S . In 6 the state at time t was expressed as $s_t = g(x_{0:t})$, where g was an embedding function and $x_{0:t} = \{x_0, x_m, \dots, x_{t-1}, x_t\}$ with x_0 being the union of user features and the sequence $x_{m:t} = \{x_m, \dots, x_{t-1}, x_t\}$ the past user-item interactions from time m to t . S was the superset that included all the possible states.

In the particular case that concerns, there is no user features. Consequently, x_0 would be empty. In addition, g will be considered as the identity function, $g(x) = x$. On the other hand, session steps can be considered as the past user-item interactions. As the objective is to predict click-outs, each state will be considered as the sequence $s_t = \{x_m, \dots, x_i, \dots, x_t, \dots, x_h\}$, where x_i is the session step at instant i , t is the last instant before the click-out, h is the observation height and m is an instant between 1 (first session step) and $t - h$.

On the other hand, and as pointed out in 2.2, in order to take profit of DRL methods, data (specially the observation) should be expressed as an image. That is, as a $h \times w \times c$ matrix with values between 0 and 255, where h , w and c are the height, the width and the number of channels, respectively. Therefore, x_i is a row and there are at most h in the state. If $t < h$, the state would be filled with $h - t$ null rows. If, on the other hand, $t > h$, the first $t - h$ user-item interactions would be removed. On the other side, w depends on the length of x_i , determined by the number of features in a step.

If a session contains more than one click-out, first sequence would be from session step 1 (or $t - h$ in case the click-out step is greater than h) until first session click-out step. Then, next state would be the past sequence plus the steps until the next click-out, including the first click-out one.

The number of columns (features), rows (interactions) and channels will be adjusted during the implementation.

7.2.2. A definition

In 6, the action a was generalized as recommending one or more items i from the item set I , that is, $a = (i_1, \dots, i_n)$, where n is the number of items recommended. Also, the action depended on the state. A was the superset that included all the possible actions.

In the case that concerns, the actual recommender system proposes 25 items before each click-out, without restricting any item depending on the state. In the case of the proposed RS, it will start by recommending 1 item to speed up training. Therefore, the action can be considered as $a = i$, being i the item recommended. The number of items present on the item set will be adjusted in the training section.

7.2.3. R definition

As stated in 6, the reward depends strongly on the application and is usually a combination of instant and delayed reward. As early mentioned, the algorithm will be in fact learning from a dataset. Therefore, the reward will be given whenever the proposed RL-based RS recommendations fit to the Trivago's ones. The idea is to start with an instant reward and depending on the learning process try to add a delayed one to see the effects in the long-term.

Regardless of whether 1 or 25 items are provided, the reward will be a sum of two parts:

- +0.1 for each of the proposals (or the only proposal) inside the set of 25 candidates provided by Trivago.
- If one of the proposals (or the only proposal) is the clicked-out item, an extra reward will be given depending on the index of that item in the 25 proposals given by Trivago. The maximum reward for this concept, obtained when the proposal is the clicked-out item, is 10.

7.3. RS Implementation

Once the problem has been described as an MDP, the objective is to implement the RL-based RS. The implementation will be divided in the following phases:

- Data preprocessing: includes the dataset reduction and the features encoding.
- Environment implementation: includes implementing the necessary code to model the dataset as an RL environment.
- Agent implementation: includes the algorithm selection and the integration with the RL environment.

7.3.1. Data preprocessing

As explained in 7.2.1, the state will be expressed as an image. Also, in 7.1.2.1 the information included in each session was detailed. The main objective of the preprocessing step is to transform the information included in each session into a numerical format in order to construct each state. Before, unnecessary information will be discarded as well as very short or very long observations, in order to simplify the training.

7.3.1.1. Dataset reduction

As stated in 7.1.2.1, the information provided by Trivago is divided in two sets: the training and the test set. Some useful stats from these sets are shown in Table 8:

Feature	Training set	Test set
Users	730.803	250.852
Interactions	15.932.992	3.782.335
Sessions	910.683	291.381
Click-out actions	1.586.586	528.779
<i>Items</i>	927.142	927.142

Table 8. Train and test sets stats

As it can be seen, the training set is composed by more than 900.000 recommendable items. This number of items makes the training unfeasible with the available resources. In fact, most of the RL examples pointed out in 4.4 have action spaces with less than 10 actions (in case of the Atari games [21], for instance). The idea then is to reduce the dataset to start with a smaller item set and increase the number of items progressively.

In order to reduce the dataset to have a reduced item set, sessions interacted less than x times

in the training set will be removed. That is, those numeric references in the column “reference” that appear less than x times. The x parameter will be fixed during the training process. In order to have the same item set in both sets, sessions containing items not interacted x times in the training set have to be removed from the test set as well.

Also, sessions with too few steps would mean lots of null rows in the observation and sparse information. In order to keep the observation richer in information, sessions with less than 10 steps in the training have been removed.

Finally, the last two columns of the dataset correspond to the impressions provided by Trivago and their prices whenever a click-out takes place. The price is probably a key element when choosing which accommodation to go on a vacation. Unfortunately, prices are not available for the whole item set at each time stamp but for a reduced subset of elements in certain moments. Moreover, putting this information in the state supposes adding too many cells but most of it empty (click-outs are a minority over the whole set of steps), and the number of cells they occupy is very high in relation to what the rest of the state does. Therefore, the price information is suppressed.

With respect to the impressions, it is true that having past proposed items in the state could be useful and in fact some papers went in this direction (as pointed out in 5.4.2). However, adding this information has the same problem than adding the prices one. Therefore, training will be done without considering this column.

7.3.1.2. Features encoding

As stated in 7.2.1, state information has to be expressed in 0-255 cells. On the other hand, most of the information provided not only is not expressed in that range, but neither in numeric format. Therefore, the objective of this section is to encode the information provided in 0-255 range cells.

First a detailed analysis of the features provided has to be done, in order to see what type they are (numeric, text, ...) and what values they can take. Excluding impressions and prices, the dataset consists of 10 features. If no session is deleted because of low-interacted reference ($x = 0$), these features keep the following data:

- User ID: 730803 unique users, expressed in alphanumeric format.
- Session ID: 1202064 unique sessions, expressed in alphanumeric format.
- Timestamp: time instants go from 1541030408 to 1541548799 (514.391 instants) in the train set and from 1541548807 to 1541721599 (172.792 instants) in the test set, expressed in numeric format.
- Step: there are at most 3.522 steps per session, expressed in numeric format.

However, 99.4% of click-outs (the step when the observation is provided to the agent) happen to be before step 255.

- Action type: the action can take up to 10 values, expressed in text format.
- Reference: references can take two kinds of values: numeric and text. Numeric references correspond to the impressions present in the item set. As opposite, text references can correspond to sort order description, filter description, destination name and POI name. There are 927.142 numeric references, while only 36.404 text ones.
- Platform: there are 55 different platforms, expressed in text format.
- City: there are 34.752 different cities, expressed in text format.
- Device: 3 types of devices are used, expressed in text format.
- Current filters: filters are a special case. In total, there are 202 filters. 155 of them are the properties that appear assigned to the accommodations in the item metadata set and the other 47 are independent. Another particularity is that it is possible to have more than one active filter at a time, dramatically increasing the values that current filters cell can take.

Once the features have been analyzed, they should be expressed in the aforementioned numerical range. Three kinds of features can be distinguished based on the data type they present:

- Numerical data. This category includes: timestamp, step and numerical references. This information can be straightforward expressed in 0-255 range. Those values greater than 255 can be expressed in binary and again in decimal but grouped by bytes, since a byte can store up to $2^8 = 256$ values (from 0 to 255, both included). For instance, the number 1541030408 (lower timestamp) would result in the tuple (91, 218, 66, 8) (see Figure 12).

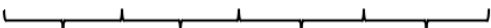
Decimal:	1541030408
Binary:	0b01011011110110100100001000001000
	
Decimal (4 bytes):	91 218 66 8

Figure 12. Numerical data encoding example

And therefore occupy 4 cells. This step is done when constructing the observation, since the bytes each feature needs varies depending on the action space.

- Text and alphanumerical data. This category includes: user ID, session ID, action type, text references, platform, city and device. In order to express this information in the desired numerical range, it is necessary first to express it as a number. One way is to encode each feature with values between 0 and n-classes. I.e., user IDs will go from 0

to 730.803-1 = 730.802. Then, they would be expressed in the same way than numerical data.

- Binary data. This category is devoted to the *current_filters* column. Since many filters can be activated at a time, the idea is to construct a binary vector where each bit corresponds to a filter and a 1 or a 0 means whether the filter is active or not. Once the binary vector is constructed, it is expressed in decimal base grouped by bytes as explained before.

7.3.2. Environment implementation

One possibility is to construct the algorithm and the necessary framework to make it work from scratch, since most RL algorithms are publicly accessible. However, any change in the state size or the action space, not to mention an algorithm change, would imply lots of work. Therefore, it is out of scope. Instead, there are some libraries designed with this purpose. Above all, Gym stands out.

Gym (<https://gym.openai.com/>) is a library created by OpenAI (<https://openai.com/>) for developing and comparing reinforcement learning algorithms. Gym was launched in 2016 with the aim of establishing a standard framework in which to express RL problems, so that environments were easy to set up and it would serve as a benchmarking tool, were published research could be reproduced and compared.

The core gym feature is the *Env*, which is the environment interface. Gym provides a large set of environments, including Atari games, control tasks or physics engines such as MuJoCo. It also allows you to create your own environment, as in the case of this project. [25]

A custom environment can be defined through the class *Env*. This class encapsulates an environment with arbitrary dynamics. The main methods that are going to be used are:

- *step(self, action)*: runs one timestep of the environment. Takes an action as an input and returns:
 - observation: the agent's observation of the environment after executing previous action.
 - reward: the reward returned after executing previous action.
 - done: whether the episode has ended.
 - info: auxiliary information provided.
- *reset(self)*: resets the environment to an initial state and returns the initial observation.

There are also a set of important attributes needed to define the environment:

- *action_space*: the space object corresponding to valid actions.
- *observation_space*: the space object corresponding to valid observations.

These two spaces should be of the class *gym.spaces*, that admits four possible type of spaces:

- *Box*: the box space consists of a box in \mathbb{R}^n . Each interval has the form of one of $\{[a, b], [a, \infty), (-\infty, b], (\infty, -\infty)\}$.
- *Discrete*: the discrete space consists of set of discrete actions.
- *MultiDiscrete*: the multi discrete space consists of a series of discrete spaces with different number of values in each of them.
- *MultiBinary*: the multi binary space consists of a n-shape binary space.
- *Tuple*: a set of simpler spaces in tuple format.
- *Dictionary*: a set of simpler spaces in dictionary format.

In order to use the Gym library, it is necessary therefore to adapt the chosen dataset to its framework.

7.3.2.1. *Env* attributes

Regarding the attributes, the following has been considered:

- *observation_space*: as pointed out in 7.2.1, the state (in fact, the observation of the state) is an *hxc* image with values compressed between 0 and 255 and where each line represents an interaction step between the user and the RS. Therefore, the space corresponding to all possible observations takes also this form. The *observation_space* will then be defined as a *Box* in \mathbb{R}^3 where each interval takes values $[0, 255]$. The size of the image depends on the steps and features considered and will be set in the training.
- *action_space*: as pointed out in 7.2.2, the action corresponds to recommending an item *i*. Therefore, the space will be modeled as a *Discrete* where the number of items depends on the fraction of the dataset used.

7.3.2.2. *Env* methods

The interaction between the *Env* class and the datasets is managed with the *ObserveDataset* class. This class is responsible for sending the *Env* class all the information it needs: the observation of the environment, the previous action reward, and whether the goal was achieved. To do so, it processes the information contained in the set in order of appearance. It can process the training and the test set.

Out of the methods that take part of the class, the following stand out:

- **Constructor**. As input, the class takes the dataset used: the train or the test. Then, the following variables are initiated:
 - *image_size*: stores the observation size. Since the observation can include

more or less information depending on the number of features included, their size and the desired steps stored, it will vary during training.

- *items*: stores the items dataset in a data structure.
 - *sessions*: stores the sessions dataset (either the train or the test one) in a data structure.
 - *sessions_pointer*: stores the row number of the first unprocessed interaction. Initial value is 0
 - *obs*: stores the observation of the environment in an array with the size of the *image_size* variable. Initial value is an *image_size* array with null values.
 - *obs_pointer*: stores the row number of the first empty row in the observation. Initial value is 0
 - *true_impressions*: stores the items (represented by their item ID) recommended by Trivago in the last observation processed in an array. Initial value is an empty array.
 - *seen_users*: stores the users (represented by their user ID) processed so far.
- *reset*. Resets *sessions_pointer*, *obs*, *obs_pointer* and *seen_users* variables to their initial values.
 - *done*. Returns whether the last *sessions* line processed was the last in the sessions structure, and therefore, all the dataset information has been transferred to the agent. Since there is not a defined goal (as one can find in a game), this how the final state is defined.
 - *reward*. Returns the last executed action reward. It takes as input the action given by the algorithm (one or more recommended items) and compares it with the impressions proposed in the dataset in the last click-out step.
 - *observation*. Returns the environment's current observation and the done signal (given by the *done* method). Before doing an observation, two situations can be presented to the agent: a new user comes in (in a new session) or the session in which the agent was continues (last click-out step was not the end of the session). In the first situation, the observation is restarted. That is because the observation of the environment encompasses only the last user-item interactions, not including what previous users did. After that, session steps are added to the observation in both situations. This addition is managed by two methods:
 - *action_to_array*: receives a session step, encodes each of its features in values from 0 to 255 and sorts them in an array whose length is equal to the number of columns in the observation (the second dimension of *image_size*).
 - *add_actions_session*: adds sessions steps to the observation until current session ends or there's a click-out action. To do so, it successively calls the *action_to_array* method in the current session step and moves on into the next one, updating both *sessions_pointer* and *obs_pointer*.

If the previous method ended because there was a click-out, both the click-out reference and the impressions given by Trivago are stored in the *true_impressions* variable. Then, the done method is called and both the observation and the done signal are returned. As opposite, If the previous method ended because there were no more steps in the session and therefore the session ended in a non-click-out action, the *observation* method is called again.

7.3.3. Agent implementation

This stage includes the algorithm selection process and the integration of the selected algorithms in the Gym interface in the training script.

7.3.3.1. Algorithm selection

As explained in 4, there are basically two main approaches to train agents in model-free RL: policy-based and Q-learning methods. Policy optimization methods tend to be more stable and reliable. Also, Stable Baselines library allows one or the other algorithm to be used depending on its support for discrete/continuous actions and the possibility of training several simultaneous agents. Only Advantage Actor Critic (A2C), Deep Q Networks (DQN), Hindsight Experience Replay (HER) and Proximal Policy Optimization (PPO) support discrete action spaces. Out of them, A2C and PPO are on-policy algorithms. Therefore, these two are going to be used to train the RS agent.

7.3.3.1.1 A2C

The A2C algorithm is derived from policy gradient methods. These methods update the policy parameters θ in the direction $\nabla_{\theta} \log \pi(a_t | s_t; \theta) R_t$, which is an unbiased estimate of $\nabla_{\theta} \mathbb{E}[R_t]$ but has a high variance. One of the ways of reducing this variance is by subtracting a learned function of the state from the return. Usually, this function is a learned estimate of the value function, $V(s; \theta_v)$. The term $R_t - V(s_t; \theta_v)$ can be then seen as the advantage of action a_t in state s_t , since R_t is an estimate of $Q(a_t, s_t)$. Therefore, the function parameters are updated in the direction given by Eq. 5.

$$\nabla_{\theta} \log \pi(a_t | s_t; \theta) (R_t - V(s_t; \theta_v)) \quad \text{Eq. 5}$$

This architecture is known as actor-critic, where the policy is seen as the actor and the value function as the critic. The actor updates the policy parameters in the direction given by the critic. Some insights of the algorithm are:

- The policy and the value function are modelled with neural networks (NN). Usually, non-output layers are shared.

- Both the policy and the value function are updated after n actions or when a terminal step is reached.
- The gradient of the full objective function also includes an entropy regularization term to improve exploration over premature convergence to local minima [36].

The pseudocode of the algorithm can be seen in Figure 13.

Algorithm 1 Advantage actor-critic - pseudocode

```

// Assume parameter vectors  $\theta$  and  $\theta_v$ 
Initialize step counter  $t \leftarrow 1$ 
Initialize episode counter  $E \leftarrow 1$ 
repeat
  Reset gradients:  $d\theta \leftarrow 0$  and  $d\theta_v \leftarrow 0$ .
   $t_{start} = t$ 
  Get state  $s_t$ 
  repeat
    Perform  $a_t$  according to policy  $\pi(a_t|s_t; \theta)$ 
    Receive reward  $r_t$  and new state  $s_{t+1}$ 
     $t \leftarrow t + 1$ 
  until terminal  $s_t$  or  $t - t_{start} == t_{max}$ 
   $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta_v) & \text{for non-terminal } s_t \text{ //Bootstrap from last state} \end{cases}$ 
  for  $i \in \{t-1, \dots, t_{start}\}$  do
     $R \leftarrow r_i + \gamma R$ 
    Accumulate gradients wrt  $\theta$ :  $d\theta \leftarrow d\theta + \nabla_{\theta} \log \pi(a_i|s_i; \theta)(R - V(s_i; \theta_v)) + \beta_e \partial H(\pi(a_i|s_i; \theta))/\partial \theta$ 
    Accumulate gradients wrt  $\theta_v$ :  $d\theta_v \leftarrow d\theta_v + \beta_v (R - V(s_i; \theta_v))(\partial V(s_i; \theta_v)/\partial \theta_v)$ 
  end for
  Perform update of  $\theta$  using  $d\theta$  and of  $\theta_v$  using  $d\theta_v$ .
   $E \leftarrow E + 1$ 
until  $E > E_{max}$ 

```

Figure 13. A2C pseudocode. Source: [36]

7.3.3.1.2 PPO

PPO is also derived from policy gradient methods. More specifically, it is derived from Trust Region Policy Optimization (TRPO). This algorithm updates the policy by taking the largest possible step that improves performance, while keeping the new and old policy close enough according to a distance measure. PPO has also some of the benefits of TRPO but is much simpler to implement.

Let $r_t(\theta)$ denote the probability ratio

$$r_t(\theta) = \frac{\pi(a_t|s_t; \theta)}{\pi(a_t|s_t; \theta_{old})} \quad \text{Eq. 6}$$

The objective function PPO proposes is

$$L(\theta) = \mathbb{E}[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)A_t)] \quad \text{Eq. 7}$$

Where the second term modifies the surrogate objective by clipping the probability ratio,

removing the incentive for moving it outside of the interval $[1 - \epsilon, 1 + \epsilon]$. Finally, it takes the minimum of the clipped and unclipped objective, so the final objective is a lower bound of the unclipped one. This way, the change in $r_t(\theta)$ is ignored when it makes the objective improve and is included when it makes it worsen. Finally, $A_t = A(s_t, a_t; \theta_{old}, \theta_v)$ is the advantage term defined in 7.3.3.1.1. The full objective function also includes an entropy regularization term to improve exploration, as with A2C. [29]

The pseudocode of the algorithm can be seen in Figure 14.

Algorithm 1 PPO-Clip

- 1: Input: initial policy parameters θ_0 , initial value function parameters ϕ_0
- 2: **for** $k = 0, 1, 2, \dots$ **do**
- 3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
- 4: Compute rewards-to-go \hat{R}_t .
- 5: Compute advantage estimates, \hat{A}_t (using any method of advantage estimation) based on the current value function V_{ϕ_k} .
- 6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg \max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \min \left(\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.

- 7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg \min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \left(V_{\phi}(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

- 8: **end for**
-

Figure 14. PPO pseudocode. Source: [33]

7.3.3.2. Algorithm integration and training script

If gym provides the environment part, Stable Baselines (SB) (<https://github.com/DLR-RM/stable-baselines3>) provides the other main element in RL: the agent. Stable Baselines is a set of implementations of RL algorithms in PyTorch (<https://pytorch.org/>), an open-source machine learning framework. It was released with the aim of having reliable implementations of RL algorithms that serve as a basis for comparing performance in different environments and against new algorithms.

One of the advantages of SB is that it relies on the OpenAI Gym interface, simplifying the algorithm integration process within the environment previously defined. This integration can be seen in the training script:

```

1. # Module import
2. from stable_baselines3 import A2C, PPO
3. from stable_baselines3.common.vec_env import *
4. from stable_baselines3.common.env_util import *
5. from stable_baselines3.common.monitor import Monitor
6. from stable_baselines3.common.callbacks import EvalCallback
7.
8. # Variables definition
9. n_items = 1342
10. train_steps = 12846
11. algorithm = PPO
12.
13. # Train environment definition
14. env = RSEnv(n_items, 'train', 1)
15.
16. # Evaluation environment and callback definition
17. eval_env = RSEnv(n_items, 'test')
18. eval_env = DummyVecEnv([lambda: env])
19. eval_env = VecTransposeImage(eval_env)
20. eval_callback = EvalCallback(eval_env, n_eval_episodes = 1, eval_freq = 512)
21.
22. # Learning process
23. model = algorithm('CnnPolicy', env, n_steps = 512, verbose = 1, seed = 1)
24. model.learn(train_steps, callback = eval_callback)

```

Figure 15. Training script

As shown in Figure 15, the integration is straightforward. One can identify five sections in the training script:

- Module import. The SB modules required for training are imported.
- Variables definition. As mentioned in 7.3.1, the action space dimension will change in order to see the performance of the agent. This variable also affects the available training steps, since a reduction in the number of items means the deletion of the sessions with low interacted items. Both variables are set here.
- Train environment definition. The custom Gym environment (defined in 7.3.2) is loaded specifying the number of items and the 'train' dataset.
- Evaluation environment and callback definition. The goal of any ML model is not to succeed in the train data but in new data. The test set serves as a proxy for this new data. Also, one of the features of SB is the use of callbacks. A callback is a set of functions that will be called at defined stages of the training process. It can be used to monitor the training, to save models, to change parameters... In this case, the predefined *EvalCallback* will be used to periodically evaluate the policy on the test set and to save the model it performs best on it.
- Learning process. The first line is devoted to the model construction. That is, the algorithm selection along with its parameters. Among all, the following stand out: policy, number of steps and seed:
 - Policy. As mentioned in 2.2, one of the objectives is to take advantage of the

CNNs when processing images. Therefore, *CnnPolicy* is selected as policy.

- Number of steps: defines the number of steps to run per update.
- Seed: specifies the seed for the pseudo random number generation. Using the same seed allows to compare trained models so that any change in performance cannot be attributed to the use of different numbers. Therefore, `seed = 1` will be used in all the training process.

The whole implementation can be found in the author's code repository. [15]

7.4. Results

The objective of this section is to train the implemented RL-based RS and analyze the results obtained.

As explained in 0, RL agents have been successful mainly when the action space was of an order of magnitude. Instead, the Trivago dataset has an action space with almost 930k possible actions if recommending one item at a time (otherwise the possibilities are nearly infinite). This requires a lot of computing power and makes the training process very time consuming. In order to accelerate the training process and taking into account that the priority is not to obtain accuracy or to solve this particular case but to demonstrate that it can be solved in this way, different reduced versions of the dataset have been used: a dataset with 524 items, another with 686 and last one with 924 items. These datasets are obtained when suppressing items interacted less than 1250, 1125 and 1000 times.

With the selected dataset sizes, the features (encoded) fit in 45 columns: 1 for the step, the action type, the platform and the device; 2 for the user ID, the session ID and the city; 4 for the timestamp; 4 for the reference and 26 for the filters. In order to work with square images, the same value has been selected for the number of steps in the observation. Indeed, the average number of steps per click-out is near 18, so there's a 150% margin. Since the first two dimensions of the matrix do not have very high values, 1 channel has been selected. Therefore, the state space is a 45x45x1 image.

In order to measure how well an agent is performing, it will be compared against the random agent. This agent (which is in fact the algorithm before starting the training) will in the test set using different seeds, as the random numbers chosen have a major impact when the data set is reduced.

In addition to training agents using RL, they have also trained using Behavioral Cloning (BC), an approach that combines RL and supervised learning.

7.4.1. Reinforcement Learning approach

If reducing the action space to 524 items, results shown in Figure 16 are obtained.



Figure 16. Episode reward obtained by different algorithms during the learning process for an action space of 524 items. RA stands for Random Agent.

Figure 16 shows the reward obtained by the agents (including the random one) in the test set at different stages of the training process in the test set. These results allow several conclusions to be drawn.

First, the agents trained through A2C and PPO outperform the random agent as long as the training is at an advanced stage. While the random agent's reward is 24,87, A2C reaches 115,4 and PPO's best models earns over 65,3.

Second; the reward increases over time. This means (along with the first conclusion) that there is a learning by the algorithm. It is common, on the other hand, to have large ups and downs in performance during training; RL is characterized by instability during training.

Third; despite the performance achieved by the models, the results are not successful at all. In a data set with almost 1000 (957) observation-action pairs and taking into account the proposed reward signal, an accuracy of 100% would mean a reward close to 10,000. It is clear then that obtaining rewards between 40-60 is not satisfactory.

Similar results are obtained when increasing the action space.

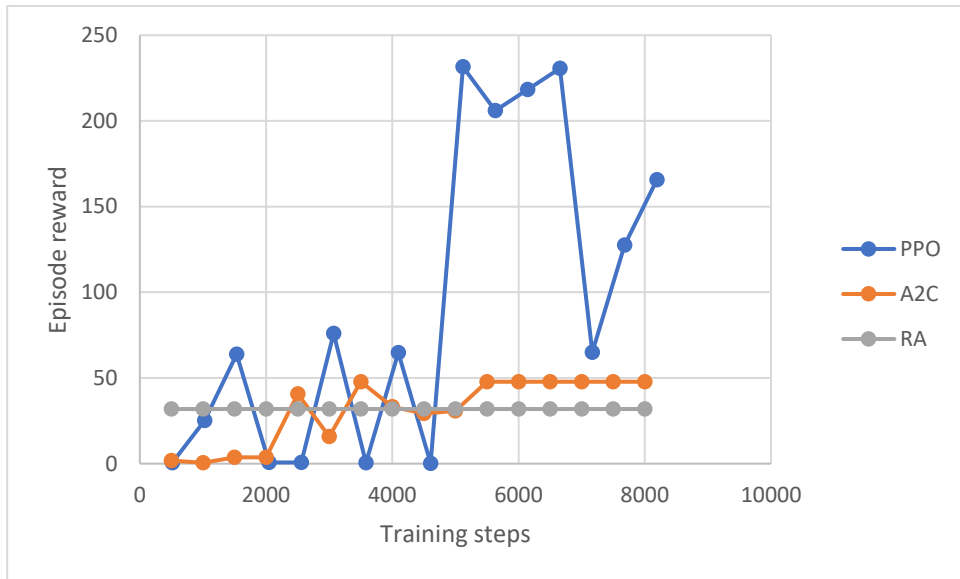


Figure 17. Episode reward obtained by different algorithms during the learning process for an action space of 686 items.

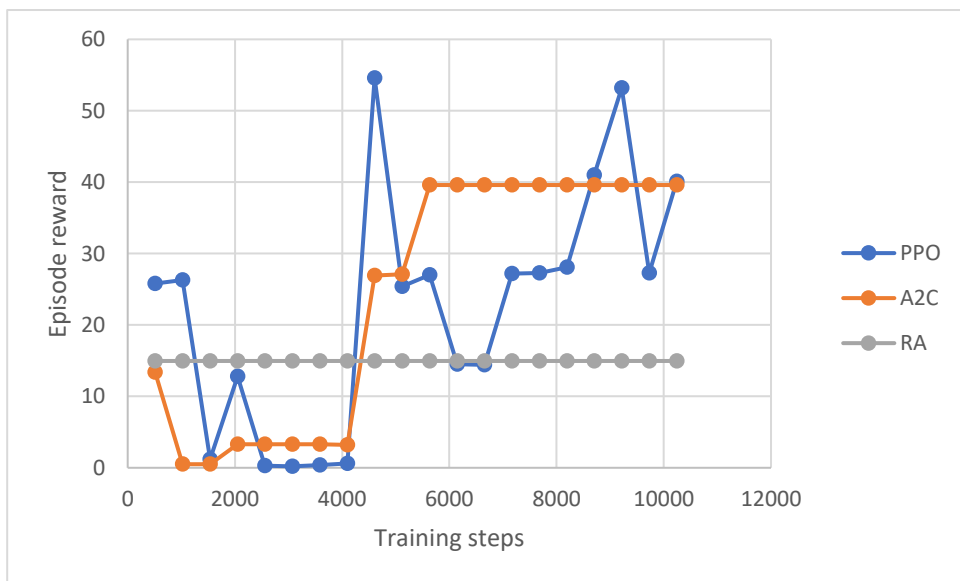


Figure 18. Episode reward obtained by different algorithms during the learning process for an action space of 924 items.

Figure 17 and Figure 18, show results for action spaces with 686 and 924 items, respectively. In the first one, PPO is the algorithm that achieves better results. While the random policy gets a reward slightly below 32, PPO reaches 231,5. On the other hand, A2C best model gets 47,8. Also, both algorithms perform better as timesteps advance.

Similar results are obtained in the second one. Leaving aside the early timesteps, highly conditioned by the initial random policy values, the trend is bullish. It is easy to observe in the



A2C algorithm, but it is also true in the PPO. Also, both algorithms end performing better than the random agent in the training last stages.

These results (greater reward achieved with the trained algorithms than with the random agent, upward trend of the reward and relatively low performance of the agent) are maintained as the action space gets bigger. Also, they are kept independently of the observation size: changes in the number of steps in the observation or the distribution of the information by altering the number of channels result in the same patrons.

The main reason for obtaining these results is the lack of data in the training set. On the one hand, the training set only contains feedback of the existing recommendations in the users' historical records, which is sparse compared with the size of the action space of the RS. Moreover, this feedback only tells which item is selected by the user, meaning one item gets a 1 and the other ones all get a 0. This feedback is clearly not enough when training from interaction. On the other hand, even if this feedback was available, it may not be enough.

The complete dataset (without reduction), for example, consists of 1.586.586 click-out actions (that is, 1,586,586 observation-action pairs) and 927.142 items. When training begins, the algorithm starts to sample actions based on a random policy: that is, the probability of recommending the correct element in the first action would be $1/927.142$. Furthermore, this first success would be obtained on average after $927.142/2 = 463.571$ attempts: almost 1/3 of the total training steps. However, this first boost would not be enough to train the policy on its own and the algorithm would continue to explore other alternatives. To the point that at the end of the training it would hit just over $1.586.586/927.142 = 1,71$. Considering accepting as valid one of the 25 items proposed by Trivago, the number of hits would rise to not much more than 42,78. This is clearly insufficient considering the agent has to map from 45x45 observations to 927.142 items. Reducing the action space by removing the less interacted items increases a bit the rate of success, but not to the point of solving this issue.

This a common problem when facing RS: the data to learn from is very sparse. And on top of that, RL is known for being sample inefficient, limited by the need to explore the state space through trial and error. That means usually millions of interactions are needed for success. In fact, if starting from a random policy it needed few interactions, it would imply getting stuck at a local minimum, which is not desirable.

Several articles speak of the difficulty of achieving good results using only RL. In [40], for instance, researchers affirm "due to the lack of sufficient data, directly generating recommendations from RL is difficult" and propose to apply their RL-based method in already built recommendation models. [19] also mentions the limitations for offline policy learning, especially when facing enormous action space. [37] also speaks about the problem of not having feedback for all possible items.

7.4.2. Behavioral Cloning approach

As seen in 7.4.1, training a RL agent from scratch for this particular case has some difficulties. One approach for overcoming these limitations is to take advantage of the fact of having Trivago recommendations in each state, a priori better than random ones, to pre-train the agent with them and then finish training through RL. This is known as Behavioral Cloning (BC) and consists of using expert demonstrations (trajectories from a controller, from another trained RL agent, ...) to pre-train RL policies as in a supervised learning problem, and therefore accelerate training. This alternative is pointed out in [40], among others.

First, the train set will be divided into a pretrain and a train set. The pretrain set will be then used to generate expert observation-action pairs in a separate dataset. After that, the algorithm policy will be extracted and pretrained through supervised learning using the previously collected trajectories. Finally, the policy will be inserted back in the RL algorithm.

For an action space of 524 items and using a 25% of the train dataset for pretraining, results before RL-training show an episode reward = 1803,7. After training through RL, the results seen in Figure 19 are obtained.

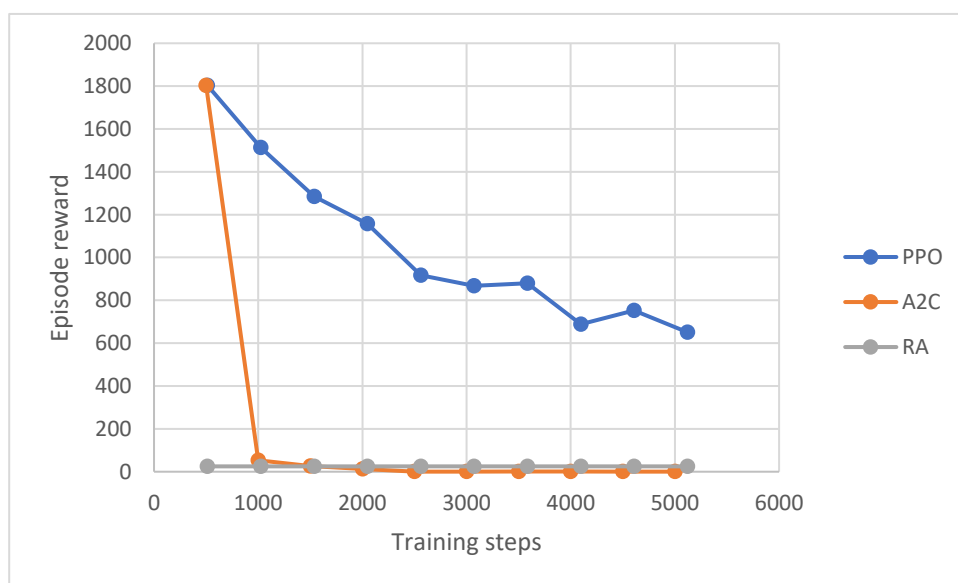


Figure 19. Evolution of episode reward during training using BC and 524 items when pretraining with the 25% of the set.

As seen in Figure 19, reward decreases dramatically as timesteps advance either with A2C or PPO.

With A2C, episode reward goes from 1803,7 to 53,4 in the first iteration. After that, reward drops continuously to 0,3. This model performs much worse than the one obtained without

pretraining (115,3). Indeed, it performs worse than the random agent. Therefore, the main conclusion is that the pretraining barely helps the RL A2C agent to get higher rewards. In fact, supervised learning alone clearly outperforms the results obtained through RL.

Better results are obtained by using PPO. Although there is also a decrease in the first timesteps, after timestep = 4000 there's a relative stabilization period where the reward is kept between 650 and 800. These models are better than the ones obtained without pretraining, where rewards went from 26,1 to 65,3 (in the stabilized period). Although one can say that pretraining before applying RL result in a better performance than without pretraining, the conclusion is the same as with the A2C algorithm: supervised learning clearly outperforms RL results.

The difference in results between the PPO and A2C algorithms is mainly due to the objective functions of each of the algorithms. As explained in 7.3.3.1.2, PPO optimizes the performance while keeping the new policy close to the old one. A2C doesn't try to keep policies close.

As explained in 7.3.3.1.2, there is a parameter in the PPO objective function to limit how far the new policy can be from the old one. This parameter is called *clip_range* and can be tuned as desired (its default value, used in Figure 19, is 0,2). Also, there is an analog parameter, called *clip_range_vf* that limits the value function parameter optimization. In this case and since the expert trajectories achieve good results, it is interesting to keep both the new policy and the new value function as close as possible to the new one to see if the policy can be improved. Figure 20 shows PPO results for different *clip_range* and *clip_range_vf* values.

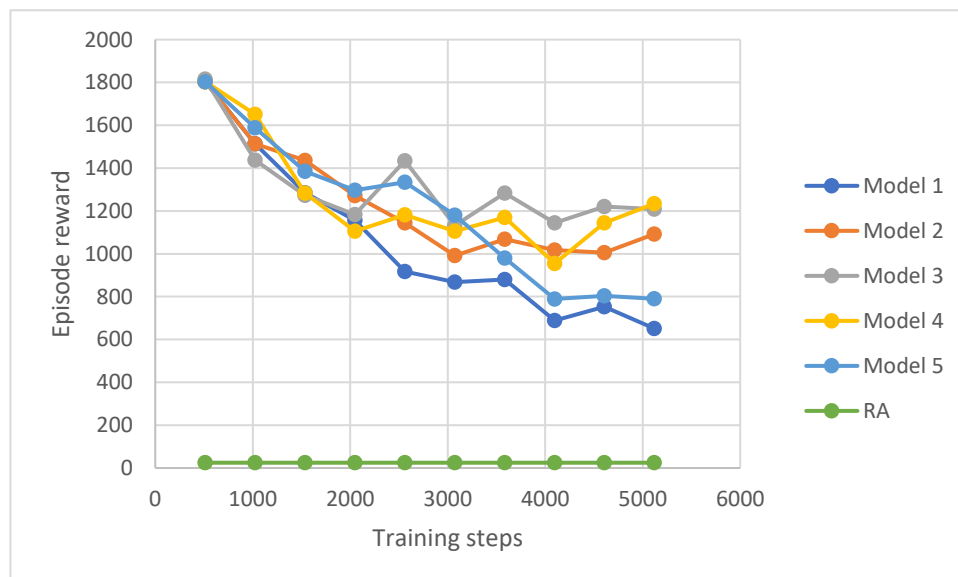


Figure 20. Evolution of episode reward during training using BC and 524 items when pretraining with 25% of the set, for some *clip_range* and *clip_range_vf* values. Model 1 has parameters *clip_range* = 0,2 and *clip_range_vf* = 0,2; Model 2: 0,2 and 0,1; Model 3: 0,1 and 0,1; Model 4: 0,1 and 0,05; Model 5:

0,05 and 0,05.

As seen in Figure 20, reducing the *clip_range* improves the model performance significantly. While when both values were 0,2 reward was stabilized over 650-800, lower values stabilize it over 1000-1200. Moreover, the stabilization phase is better defined and starts earlier (timestep = 3000). The best model is Model 4, which reaches a reward $r = 1233,5$. On the other hand, reducing *clip_range* value below 0,1 does not bring any benefit. However, this improvement does not solve the problem and supervised learning alone continues to perform better alone.

Similar results are obtained when dedicating different percentages to the pretraining phase.



Figure 21. Evolution of episode reward during training using BC and 524 items when pretraining with the 5% of the set. PPO hyperparameters are the optimal: *clip_range* = 0,1 and *clip_range_vf* = 0,05.

Figure 21 shows results when dedicating only 5% of the dataset to the pretraining phase. As it can be seen, the performance also falls in the first steps and RL doesn't improve the results obtained in the first stage. The performance loss is a bit higher compared to when pretraining with 25% of the set for the PPO algorithm (now is up to 60% compared to the 30% in Figure 212). In the case of A2C, the performance loss is lower (90% compared to almost 100% in Figure 20). However, the final performance is below the one obtained without pretraining (43,1 compared to 47,8).

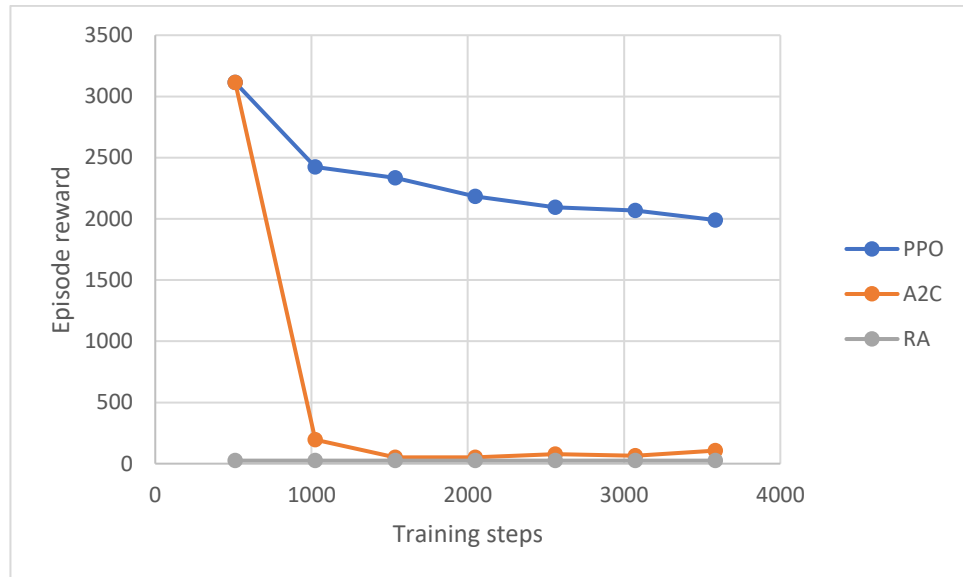


Figure 22. Evolution of episode reward during training using BC and 524 items when pretraining with the 50% of the set. PPO hyperparameters are the optimal: $clip_range = 0,1$ and $clip_range_vf = 0,05$.

As Figure 22 shows, increasing the percentage of steps dedicated to the pretraining phase does not solve the performance decrease. Although both algorithms finish the training with better results than the random agent, the decrease in performance is still notable. Increasing more the percentage of pretraining steps does not leave enough train steps in order for the performance to be stabilized. Anyway, it the same patron is reproduced.

[23] says that it is unclear how to effectively use BC with RL, but points out that it may be because the two approaches are optimizing different objective functions. To address it, they propose a combined loss function that consists of an expert BC loss together with the losses of the other RL objective functions. The BC loss drives the actor's actions (they propose an actor-critic architecture) towards expert trajectories by minimizing the actions predicted by the policy network and the expert ones.

In a similar line, [27] affirms that this kind of forgetting when using RL is often caused by the entropy bonus present in the objective function, which encourages the agent to take actions randomly. Since those actions would not yield relevant rewards, pre-trained behavior would be erased. Instead, he proposes substituting the entropy bonus by a bonus that would keep those behaviors intact.

On the other hand, the successful results obtained by using supervised learning satisfy in part one of the premises of the work. Leaving aside the part of interaction inherent to the RL, it is possible to learn (and obtain good results) from transforming the information into images and finding patterns on them.

8. Conclusions and further work

RL-based RS stand out for: continuously updating recommendation strategies according to the evolution of user preferences and learning strategies to maximize long-term versus the short-term benefit.

This project contributes to those systems mainly in two aspects. First, it proposes a general formulation of the Markov Decision Process for the RS case. The proposed formulation encompasses several formulations previously used by other researchers and facilitates the way to later apply RL.

Second, it proposes to express the environment state as an image. This allows to take advantage of the full potential of the DRL algorithms. These algorithms work with CNN architectures, which can capture additional information by considering the positioning of the input values.

Finally, it shows the application of these two proposals in a practical case. First, it describes the particular case according to the MDP generalization proposed. Then, it converts the information in the dataset into a numerical format to later on express the state as an image. Finally, it trains a reduced version of the system through RL. Results with reduced versions of the dataset show how the trained agent offers better recommendations than the arbitrary choice. However, the system does not achieve a great performance mainly due to the lack of interactions. This can be caused in part due to the lack of feedback to not recommended items by the original RS.

As mentioned, training a RS using RL from offline data is not straightforward. RL is sample inefficient and needs lots (millions) of interactions, especially when the action space is huge. That's why it has been successful specially with low dimension action spaces and in environments such as videogames, where lots of interactions are available. As opposite, RS usually have high dimension action spaces and limited data is available.

In order to succeed in RS by using RL, two possibilities stand out above the others: increase the available interactions by constructing a simulator based on previous logged data or pretrain the policy offline not from interactions but from logged data directly.

[37] goes in the first direction. They propose an online user-agent interaction environment simulator that predicts a reward based on collaborative filtering techniques using the cosine function as similarity measure. [19], for instance, introduces a multi-head NN that models the environment and generates user responses. [8] and [38] also built a simulator.

On the other hand, [40] directly takes user-item interaction pairs to train their proposed

methods SQN and SAC. In fact, they work can also be considered as a form of imitation learning, since part of their model is trained to imitate user actions. [13] also uses logged data to pretrain the policy and later on finish the training in an online mode.

In the second approach was also included BC, whose objective was to pretrain the policy apart from expert demonstrations to later on update the policy by means of RL. However, its application is not easy since BC and RL try to optimize different functions. That leads to a decrease of performance in the second training stage.

Further work can go in these two directions.

Bibliography

- [1] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser. *Learning Synergies between Pushing and Grasping with Self Supervised Deep Reinforcement Learning*. IEEE International Conference on Intelligent Robots and Systems, pp. 4238–4245, 2018
- [2] Álvaro Guarner Escribano. 2020. *Using GANs (Generative Adversarial Networks) to generate fake patients*.
- [3] Baptiste Rocca, Joseph Rocca. 2019. *Introduction to recommender systems*. Towards data science. <https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada>
- [4] Barry Schwartz. 2015. *The Paradox of Choice: Why More Is Less*. Harper Perennial, New York, NY.
- [5] Carlos A. Gomez-Uribe and Neil Hunt. 2015. *The Netflix recommender system: Algorithms, business value, and innovation*. ACM Trans. Manage. Inf. Syst. 6, 4, Article 13 (December 2015), 19 pages. DOI: <http://dx.doi.org/10.1145/2843948>
- [6] David Valero Masachs. 2021. *Sistema recomanador per la formació d'equips utilitzant reinforcement learning*.
- [7] Esteban Piacentino. 2019. *Generative Adversarial Network based machine for fake data generation*.
- [8] 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 recommender sets*. In Proceedings of the Twenty-eighth International Joint Conference on Artificial Intelligence (IJCAI-19), Macau, 2019.
- [9] Ferran Velasco Olivera. 2021. *Resolució d'un problema de planificació industrial mitjançant reinforcement learning*.
- [10] Gerald Tesauro. *Temporal Difference Learning and TD-Gammon*. Communications of the ACM, 38(3):58–68, 1995.
- [11] Google Developers. 2021-02-01. *Recommendation Systems*. <https://developers.google.com/machine-learning/recommender>
- [12] Google Research. *What is Colab?* <https://colab.research.google.com/>

- [13] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. 2018. DRN: A Deep Reinforcement Learning Framework for News Recommender. In WWW 2018: The 2018 Web Conference, April 23–27, 2018, Lyon, France. ACM, New York, NY, USA 10 Pages. <https://doi.org/10.1145/3178876.3185994>
- [14] Guy Shani, David Heckerman, and Ronen I Brafman. 2005. An MDP-based recommender system. *Journal of Machine Learning Research* 6, Sep (2005), 1265-1295.
- [15] Héctor Izquierdo. 2021. RL-based RS. <https://github.com/hectorizquierdo/Reinforcement-learning-based-recommender-system>
- [16] Ian MacKenzie, Chris Meyer, and Steve Noble. (October 1, 2013). *How retailers can keep up with the consumers*. McKinsey. <https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers>
- [17] Khalifa, Nour Eldeen & Taha, Mohamed & Ezzat, Dalia & Slowik, Adam & Hassanien, Aboul Ella. (2020). *Artificial Intelligence Technique for Gene Expression by Tumor RNA-Seq Data: A Novel Optimized Deep Learning Approach*. *IEEE Access*. 8. 22874-22883. 10.1109/ACCESS.2020.2970210.
- [18] LinkedIn. *Discover your earning potential*. Recovered on August 2021 from <https://www.linkedin.com/salary/>
- [19] Lixin Zou, Long Xia, Zhuoye Ding, Jiaying Song, Weidong Liu, and Dawei Yin. 2019. Reinforcement Learning to Optimize Long-term User Engagement in Recommender Systems. In The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '19), August 4–8, 2019, Anchorage, AK, USA. ACM, New York, NY, USA, 9 pages. <https://doi.org/10.1145/3292500.3330668>
- [20] Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed H Chi. *Top-k off-policy correction for a reinforce recommender system*. In Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, pages 456–464, 2019.
- [21] Mnih, Volodymyr & Kavukcuoglu, Koray & Silver, David & Graves, Alex & Antonoglou, Ioannis & Wierstra, Daan & Riedmiller, Martin. (2013). *Playing Atari with Deep Reinforcement Learning*.
- [22] Mnih, Volodymyr & Kavukcuoglu, Koray & Silver, David & Rusu, Andrei & Veness, Joel & Bellemare, Marc & Graves, Alex & Riedmiller, Martin & Fidjeland, Andreas & Ostrovski, Georg & Petersen, Stig & Beattie, Charles & Sadik, Amir & Antonoglou, Ioannis & King,

- Helen & Kumaran, Dharshan & Wierstra, Daan & Legg, Shane & Hassabis, Demis. (2015). *Human-level control through deep reinforcement learning*. *Nature*. 518. 529-33. 10.1038/nature14236.
- [23] Nair, Ashvin & Dalal, Murtaza & Gupta, Abhishek & Levine, Sergey. (2020). *Accelerating Online Reinforcement Learning with Offline Datasets*.
- [24] NVIDIA Developer. *Convolutional Neural Network (CNN)*.
<https://developer.nvidia.com/discover/convolutional-neural-network>
- [25] OpenAI Gym. *Background: why Gym? (2016)*. <https://gym.openai.com/docs/>
- [26] Paul Covington, Jay Adams, Emre Sargin. *Deep Neural Networks for YouTube Recommendations*. In Proceedings of the 10th ACM Conference on Recommender Systems, ACM, New York, NY, USA (2016)
- [27] Pickled ML. 2019. *Competing in the Obstacle Tower Challenge*
<https://blog.aqnichol.com/2019/07/24/competing-in-the-obstacle-tower-challenge/>
- [28] RecSys. *Trivago RecSys Challenge 2019 Dataset*.
<https://recsys.trivago.cloud/challenge/dataset/>
- [29] Schulman, John & Wolski, Filip & Dhariwal, Prafulla & Radford, Alec & Klimov, Oleg. (2017). *Proximal Policy Optimization Algorithms*.
- [30] Sharma, A., Vans, E., Shigemizu, D. et al. *DeepInsight: A methodology to transform a non-image data to an image for convolution neural network architecture*. *Sci Rep* 9, 11399 (2019). <https://doi.org/10.1038/s41598-019-47765-6>
- [31] Silver D, Hubert T, Schrittwieser J, Antonoglou I, Lai M, Guez A, et al. *A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play*. *Science*. 2018; 362(6419):1140–1144.
- [32] Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, et al. *Mastering the game of Go without human knowledge*. *Nature*. 2017; 550(7676):354–359.
- [33] Stable Baselines. *PPO*. <https://spinningup.openai.com/en/latest/algorithms/ppo.html>
- [34] TensorFlow. 2021. *Introduction to RL and Deep Q Networks*.
https://www.tensorflow.org/agents/tutorials/0_intro_rl
- [35] UPC. *Taules retributives del personal docent i investigador. Any 2020* [PDF file]
Recovered on August 2021 from [https://www.upc.edu/transparencia/ca/informacio-de-](https://www.upc.edu/transparencia/ca/informacio-de)

personal/

- [36] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, Koray Kavukcuoglu. *Asynchronous methods for Deep Reinforcement Learning*. Proceedings of the 33rd International Conference on Machine Learning, PMLR 48:1928-1937, 2016.
- [37] Xiangyu Zhao, Liang Zhang, Long Xia, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2019. *Deep Reinforcement Learning for List-wise Recommenders*. In The 1st Workshop on Deep Reinforcement Learning for Knowledge Discovery (DRL4KDD '19), August 5, 2019, Anchorage, AK, USA. ACM, New York, NY, USA, 9 pages.
- [38] Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Long Xia, Jiliang Tang, and Dawei Yin. 2018. *Recommenders with Negative Feedback via Pairwise Deep Reinforcement Learning*. In KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom. ACM, New York, NY, USA, 9 pages.
- [39] Xiangyu Zhao, Long Xia, Liang Zhang, Zhuoye Ding, Dawei Yin, and Jiliang Tang. 2018. *Deep Reinforcement Learning for Page-wise Recommenders*. In Proceedings of the 12th ACM Recommender Systems Conference. ACM, 95–103.
- [40] Xin Xin, Alexandros Karatzoglou, Ioannis Arapakis, and Joemon M. Jose. 2020. *Self-Supervised Reinforcement Learning for Recommender Systems*. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '20), July 25–30, 2020, Virtual Event, China. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3397271.3401147>
- [41] Xinshi Chen, Shuang Chen Li, Hui Li, Shaohua Jiang, Yuan Qi, and Le Song. *Generative adversarial user model for reinforcement learning based recommender system*. In ICML, 2019.
- [42] Xueying Bai, Jian Guan, Hongning Wang. 2019. *Model-Based Reinforcement Learning with Adversarial Training for Online Recommender*. In the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.
- [43] Yanan Wang, Yong Ge, Li Li, Rui Chen and Tong Xu. *M³Rec: An Offline Meta-Level Model-based Reinforcement Learning Approach for Cold-Start Recommender*.