A Thesis Title

Hernan Santiago Gonzalez Toral

Department of Computer Science

University College London

hernangt12re3@gmail.com

Supervisor

Dr. Jun Wang

jun.wang.l@cs.ucl.ac.uk

This report is submitted as part requirement for the

MSc in Web Science & Big Data Analytics

at

University College London.

It is substantially the result of my own work except

where explicitly indicated in the text.

Department of Computer Science

University College London

August 3, 2016

This page is purposely left blank.

Abstract

My research is about stuff.

It begins with a study of some stuff, and then some other stuff and things.

There is a 300-word limit on your abstract.

This side is purposely left blank.

Acknowledgements

Acknowledge all the things!

This side is purposely left blank.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Overview

The volume of information available on the internet have been increasing rapidly with the explosive growth of the World Wide Web, e-Commerce and other online services, making difficult for users to find relevant products in a short period of time. To avoid this problem, many websites use recommendation systems to help customers to find items that satisfies their needs[***](Recommender Systems; Resnick). Suggestions for books on Amazon, or movies on Netflix, are real-world examples of Recommender Systems (RS) that demonstrate that the evolution of (RS) and the web should go hand-in-hand. The final goal of a RS is to generate meaningful recommendations to a collection of users for items that might interest them.

The design of such recommendation engines depend on the domain and the particular characteristics of the data available. *Collaborative Filtering* systems analyze historical interactions alone, while *Content-based* RS systems are based on user profile attributes; and *hybrid* techniques combine both of these designs. Latest studies in CF [***](Recommender Systems survey) showed that combining conceptual and usage information can improve the quality of web recommendation.

Current RS typically act in a greedy manner by recommending items with the highest user ratings. However, greedy recommendations are suboptimal over the long term. This approach does not actively gather information on user preferences and fails to recommend novel songs that are potentially interesting. A successful recommender system must balance the need to explore user preferences and to exploit this information for recommendation.

Therefore, the architecture of recommender systems and their evaluation on real-world problems is an active area of research. In fact, RS have found an active application area for diverse Information Retrieval, Web Mining and Machine Learning techniques. Reinforcement learning[***](Reinforcement Learning: an Introduction) for example, has been suited to solve the recommendation problem. In general, once a user makes her choice based on a list of recommendations given by a RS, a new list of recommended items is then presented. Thus, the recommendation process can be thought as a sequential process as in many domains, user choices are sequential by nature (e.g. a user buy a book by the author of a recent book we liked).

Indeed, reinforcement learning has been demonstrating to be potentially effective approach in the domain and particularly good to solve the cold-start problem and in some extent, the exploration/exploitation tradeoff[***](An MDP-based Recommender System). Nevertheless, it has received relatively little attention and found only limited application.

On the other hand, most recent research, like [***](Collaborative Deep Learning for Recommender Systems), have focused on making a generalization of recent advances in Deep Learning[***](Representation Learning: A Review and New Perspectives) from Identically and Independently Distributed (i.i.d.) input to the non-i.i.d. (CF-based) input, and propose hierarchical Bayesian models that jointly performs deep representation learning for the content information and CF for estimating the ratings matrix. Experiments carried out shown that the Deep Learning approaches can significantly outperform the state of the art. As a result, current research has been getting involved on using deep representational approaches that can lead to better recommendations.

Furthermore, a recent work presented by Dulac-Arnold et al. on deep reinforcement learning in large discrete action spaces[***] presented a new policy architecture that not only allows reinforcement learning methods to be applied to large-scale learning problems, and to operate efficiently with a large number of actions, but also can generalize over the action set in logarithmic time. This algorithm showed to converge under a simulated recommender system environment with a large number of songs.

1.2 Dissertation Objectives and Structure

Chapter 2 introduces some definitions of Recommender Systems and Reinforcement Learning, Later, it reviews some related work that has been made for modeling recommender systems using reinforcement learning approaches and deep learning. Chapter 3 defines the proposed model and the implementation details, while Chapter 4 details the information about experiments carried out and evaluation of results. Finally, we present a summary of lessons learned, conclusions and future work of the thesis project.

Chapter 2

Background

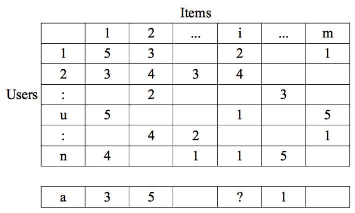
2.1 Recommender Systems

**References to papers are temporarily described with their title in order to facilitate the identification of the source, but they will be replaced later with proper latex references

In recent years, there has been growing focus on the study of Recommender Systems (RSs) where different techniques have been published in order to to address the problem of information overload on the Internet. Consequently, the evolution of RS and the web usually go hand-in hand. Resnick and Varian[***] defined a RS as a system that help users limit their search by supplying a list of items that might interest a specific user. It has found an active application area for diverse Information Retrieval, Web Mining and Machine Learning techniques that help to solve typical recommendation problems problems. Additionally, different RS mechanisms have been categorized depending on the way they analyze the data sources to find affinities between users and items.

The most general setting in which recommender systems are studied is presented in Figure 2.1. Having a rating matrix of n users and m items representing the current user preferences, the task of a RS is to predict all missing ratings $r_{a,i}$ for the active user a, and then recommend the item(s) with the highest rate[***](Recommender Systems). However the user ratings matrix is typically sparse, as most users do not rate most items. Different approaches to solve this task can be categorized into the following types:

- *Collaborative Filtering (CF):* in CF systems, a user is recommended items based on the past ratings of all users collectively
- *Content-based recommending(CB):* recommend items that are similar in content to items the user had liked in the past, or matched to pre-defined attributes of the user.
- *Hybrid approaches:* these methods combine both CF and CB approaches.



		Items					
		1	2	...	i	...	m
	1	5	3		2		1
	2	3	4	3	4		
Users	:		2			3	
	u	5			1		5
	:		4	2			1
	n	4		1	1	5	
	a	3	5		?	1	

Figure 2.1: Typical user ratings matrix, where each cell $r_{u,i}$ corresponds to the rating of user u for item i.

2.1.1 Collaborative Filtering

CF systems work by collecting user feedback in the form of ratings for items and exploiting similarities amongst several users. To build a user?s neighborhood, these methods initially rely on a database of past users interactions which can be stored in the form of explicit ratings (e.g., 5 stars to a great movie, 1 star to a bad one), or implicit ratings (e.g. a song played by a user, or an item bought by her). In general, CF methods are subdivided into memory-based and model-based approaches.

Memory-based CF

Items are recommended to an active user using a weighted combination of ratings in a subset of users chosen based on their rating similarity to the target user. The most commonly used measure of similarity is the Pearson correlation coefficient[***] (GroupLens: An open architecture for collaborative fil- tering of netnews) which measures the extent to which there is a linear dependence between the ratings of two users. Alternatively, the ratings of two users can be represented as a vector in an m-dimensional space and compute their similarity based on the cosine similarity between them. Previous studies[***] (Empirical analysis of predictive algorithms for collaborative filtering) found that the former generally performs better for predicting user recommendations.

These Memory-based CF methods have been extended and improved over the years. Linden, Smith, and York[***](Amazon.com recommendations: Item-to-item collaborative filtering) proposed an *Item-based CF* (or item-item CF) method to overcome the problem of dimensionality found when conventional memory-based algorithms are applied find similarities between millions of users and items and thus cannot scale well. This approach, which matches a user?s rated items using Pearson correlation of items, lead to faster online systems and improved recommendations (Linden et al., 2003)[***]

Model-based CF

Model-based CF provides recommendations by estimating parameters using statistical models for user ratings. The most widely used models are Bayesian classifiers, neural networks, fuzzy systems, genetic algorithms, latent features and matrix factorization[***]Recommender Systems survey. For instance, the CF can be turned into a classification problem, where a classifier is built for each active user representing items as features over users and available ratings as labels. However, latent factor and matrix factorization models have emerged as a state-of-the-art methodology in this class of techniques[***] (Matrix factorization techniques for recommender system).

Matrix Factorization

To reduce the high levels of sparsity in RS databases, matrix factorization in conjunction with dimensionality reduction techniques have been used. Having a sets U of users, and a set D of items, let $\mathbf{R}$ of size $|U| \times |D|$ be the matrix that contains all the ratings that the users have assigned to the items. Assuming that we would like to discover K latent features, the task is to find two matrices $\mathbf{P}$ (a $|U| \times K$ matrix) and $\mathbf{Q}$ (a $|D| \times K$ matrix) such that their product approximates $\mathbf{R}$:

$$\mathbf{R} \approx \mathbf{P} \times \mathbf{Q^T} = \hat{\mathbf{R}} \tag{2.1}$$

Each row of $\mathbf{P}$ would represent the strength of the associations between a user and the features. Similarly, each row of $\mathbf{Q}$ would represent the strength of the associations between an item and the features. One way to obtain $\mathbf{P}$ and $\mathbf{Q}$ is to first initialize both matrices with some values, and calculate how different their product is to $\mathbf{M}$ by using gradient descent with a regularization term in order to minimize the difference between $\mathbf{R}$ and the predicted preference matrix $\hat{\mathbf{R}}$ (e.g using the Mean Square Error). The update rules and the cost function are defined as follows:

$$p'_{ik} = p_{ik} + \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + \alpha(2e_{ij}q_{kj} - \beta p_{ik}) \tag{2.2}$$

$$q'_{kj} = q_{kj} + \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + \alpha(2e_{ij}p_{ik} - \beta p_{kj}) \tag{2.3}$$

$$e_{ij}^2 = (r_{ij} - r\hat{i}j)^2 = (r_{ij} - \sum_{k=1}^{K} p'_{ik}q'_{kj})^2 \tag{2.4}$$

Another model-based technique combines Latent Semantic Index (LSI) and the reduction method Singular Value Decomposition (SVD) are typically to solve the same problem[***](Using singular value decomposition approximation for collaborative filtering). Although SVD methods provide good prediction results, it is computationally very expensive, therefore it might be only efficient in static off-line settings.

2.1.2 Content-based RS

This method compared to pure CF that only utilizes the user rating matrix, tries to make a better personalized recommendation by exploiting the knowledge about a user, such as demographic information, or about the properties of item (e.g. genre of a movie). Several approaches have treated this problem as an information retrieval (IR) task, where the content associated with the user?s preferences is treated as a query, and the unrated items are scored with relevance/similarity to this query[***](Fab: Content-based, collaborative recommendation.). Alternatively, CB-RS has also been treated as a classification task using algorithms such as k-Nearest Neighbors (k-NN), decision trees, and neural networks[***](Learning and revising user profiles: The identification of interesting web sites. Machine Learning)

2.1.3 Hybrid approaches

In order to leverage the strengths of both CB-RS and CF, there have been several hybrid approaches proposed. For instance, Melville et al. in [***](Content-boosted collaborative filtering for improved recommendations) presented a general framework for content-boosted collaborative filtering, where content-based predictions are applied to convert a sparse user ratings matrix into a full ratings matrix, and then a CF method is used to provide recommendations. In essence, this approach has been shown to perform better than pure CF, pure CB-RS, and a linear combination of the two.

2.1.4 The cold-start problem

The cold-start problem is a common issue that happens in recommender systems when it is not possible to make reliable recommendations due to an initial lack of ratings. There are three kinds of known cold-start problems: new community, new item and new user[***](Recommender Systems survey). However, the latter represents one of the greatest difficulties faced by an RS in operation. Since new users have not yet provided any rating in the RS, they cannot receive any personalized recommendations based on memory-based CF. Usually, when the users enter their firsts ratings they expect the RS to offer them personalized recommendations, but there are not enough ratings yet to be able to make reliable CF-based recommendations. As a consequence, new users feel that the RS does not offer the service they expected.

This problem is often faced using hybrid approaches such as CF-content based RS, CF-demographic based RS, or CF-social based RS[***](Recommender Systems survey.) Those methods can also be combined with clustering techniques over items in order to improve the prediction quality.

2.1.5 Trends in Recommendation Systems

Latest studies in CF showed that combining conceptual (explicit) and usage (implicit) information can improve the quality of web recommendations. Bobadilla et al.[***] presents a taxonomy for RS which unifies the current recommender methods and algorithms that can be applied to incorporate memory-based, social and content-based information into their CF system depending on the information type available. Additionally, the taxonomy depicted in Figure 2.2, incorporates at its higher levels the current evaluation methods for RS in terms of quality measures, diversity and novelty.

Moreover, Bobadilla et al. argue in [***](Recommender Systems Survey) that the most widely used algorithm for CF is the kNN. In general, kNN executes the following three tasks to generate recommendations for an active user a: (1) determine k users neighbors; (2) implement an aggregation approach with the ratings for the neighborhood in items not

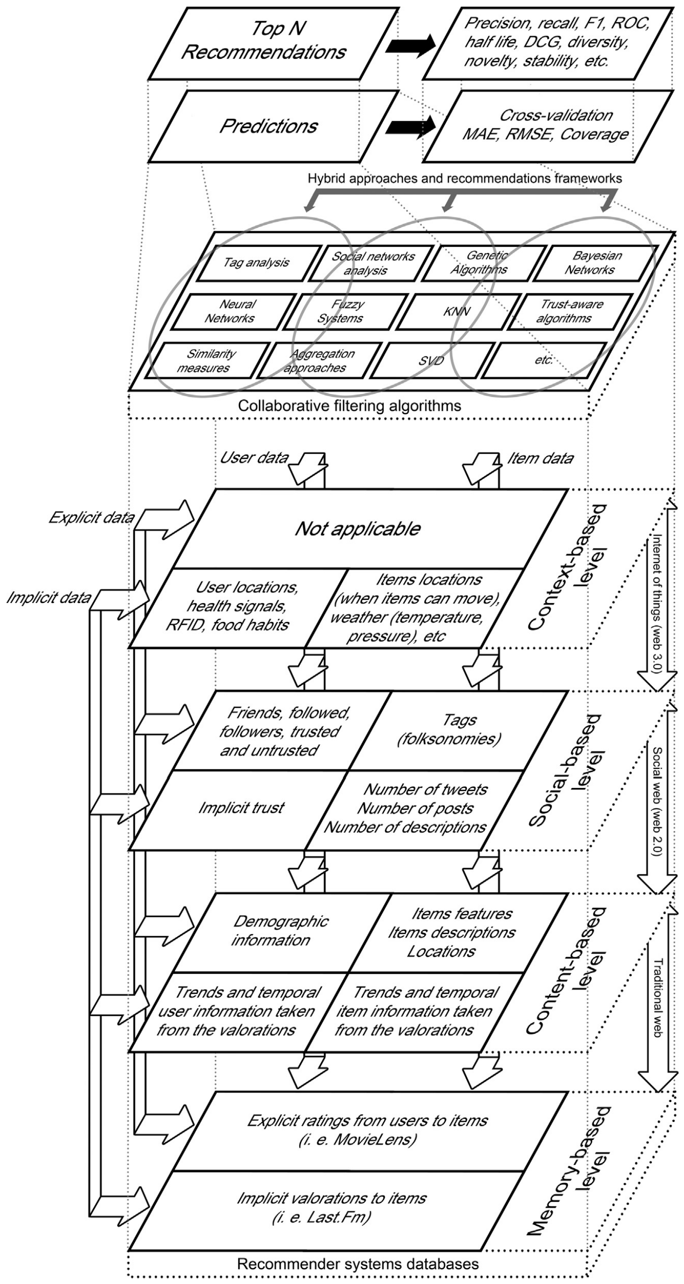

Top N Recommendations
Precision, recall, F1, ROC, half life, DCG, diversity, novelty, stability, etc.
Predictions
Cross-validation MAE, RMSE, Coverage
Hybrid approaches and recommendations frameworks
Tag analysis
Social networks analysis
Genetic Algorithms
Bayesian Networks
Neural Networks
Fuzzy Systems
KNN
Trust-aware algorithms
Similarity measures
Aggregation approaches
SVD
etc.
Collaborative filtering algorithms
User data
Item data
Explicit data
Implicit data
Not applicable
User locations, health signals, RFID, food habits
Items locations (when items can move), weather (temperature, pressure), etc
Context-based level
Internet of things (web 3.0)
Friends, followed, followers, trusted and untrusted
Tags (folksonomies)
Implicit trust
Number of tweets Number of posts Number of descriptions
Social-based level
Social web (web 2.0)
Demographic information
Items features Items descriptions Locations
Trends and temporal user information taken from the valorations
Trends and temporal item information taken from the valorations
Content-based level
Traditional web
Explicit ratings from users to items (i. e. MovieLens)
Implicit valorations to items (i. e. Last.Fm)
Memory-based level
Recommender systems databases

Figure 2.2: Recommender System taxonomy. Source: Bobadilla et al. [***](Recommender Systems survey)

rated by a; and (3) select the top N recommendations from predictions obtained in step 2.

Most recent research, like [***](Collaborative Deep Learning for Recommender Systems), make a generalization of recent advances in Deep Learning[***](Representation Learning: A Review and New Perspectives) from Identically and Independently Distributed (i.i.d.) input and generalize it to the non-i.i.d. (CF-based) input, and propose hierarchical Bayesian models that jointly performs deep representation learning for the content information and CF for the ratings matrix. On the other hand, [***](Collaborative Denoising Auto-Encoders for Top-N Recommender Systems) propose a Collaborative Deep Auto Encoders (CDAE) model which formulates the top-N recommendation problem using the Denoising Auto-Encoder framework to learn the latent factors from corrupted inputs. Experiments carried out over different domains have shown that the two Deep Learning approaches can significantly outperform the state of the art. Therefore, current research has been focusing on using deep representational approaches that can lead to better recommendations.

2.1.6 A sequential nature of the recommender process

Recommender systems usually work in a sequential manner: the RS suggest items to the user who can then accept one of the recommendations. At the next stage a new list of recommended items is calculated based on user feedback and then presented to the user. This sequential nature allows the reformulation of the recommendation process as a sequential optimization process. Furthermor, optimal recommendations do not depend only on previous items purchased, but also in the order in which those items are purchased. Zimdars et al.[***](Using temporal data for making recommendations) suggested the use of an a k-order Markov chain model (with k=3) to represent this behavior by dividing a sequence of transactions $X_1, ..., X_T$ into cases $(X_{t?k}, ..., X_{t?1}, X_t)$ for $t = 1, ..., T$. They then built a model to predict the item at the time step t given the other columns. Therefore, Recommender Systems can be modeled using reinforcement learning approaches.

2.2 Reinforcement Learning

[***](Reinforcement Learning: a Survey)(Reinforcement Learning: an Introduction)

Reinforcement learning (RL) [***] is a family of machine learning algorithms that optimize sequential decision making processes based on scalar evaluations or rewards. Similarly to an n-armed bandit model[***](The multi-armed bandit problem: decomposition and computation), RL considers the problem as a goal-directed agent interacting with an uncertain environment, and where the main objective is to perform actions that maximize the expected sum of future reward for each state in the long term. The most important feature distinguishing RL from other types of learning is that it uses training information that evaluates the actions taken rather than instructs by giving correct actions (like supervised learning algorithms). In other words, an agent must be able to learn from its own experience, like a trial-and-error search.

RL uses a formal framework[***](2) which defines the continuous interaction in terms of states, actions, and rewards, between the learner and decision-maker is called the *agent* and an *environment*. At each time step t, the agent receives some representation of the environment's state $s_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of possible states, and then selects an action $a_t \in \mathcal{A}(s_t)$, where $\mathcal{A}(s_t)$ is the set of actions available in state. One time step later, the agent receives a numerical reward $r_{t+1} \in \mathcal{R}$, and the environment turns into new state s_{t+1}. Figure **??** depicts the whole interaction in the agent-environment interface.

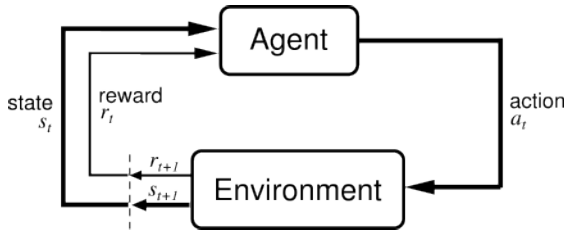


Figure 2.3: The Agent-Environment interface in a Reinforcement Learning problem. Source[***](2)

This framework is intended to be a simple way of representing four essential features of the artificial intelligence problem: (1) a *policy* (commonly stochastic) defines a mapping from the perceived states of the environment to actions to be taken when in those states; (2) a *reward* function maps each state-action pair to a single number that represents how good or bad the new state is for the environment; (3) a *value* function specifies the total amount of reward an agent can expect to accumulate over the future and starting from a given state. This function is considered the most importatn as it is used by the agent during decision-making and planning; and optionally (4) a model that mimics the behavior of the environment (transitions and rewards) and provide way of deciding which action to perform considering possible future situations before they are actually experienced.

In general, agents are categorized depending on how the reinforcement learning problem needs to be modeled. *Value-based* and *Policy-based* agents are solely based on the value and policy functions respectively. *Actor-critic* agents use both policy and value functions; *Model-free* agents use policy and/or value function but no model; and *Model-based* agents have all properties mentioned above.

On the other hand, The *return* R_t is defined as the sum of the discounted future rewards over an episode of T time steps that an agent actually seeks to maximize:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.5}$$

where $\gamma, 0 \leq \gamma \leq 1$ is a discount factor that determines the present value of future rewards

2.2.1 Markov Decision Processes

A Markov Decision Process (MDP) is a model for sequential stochastic decision problems. As such, it is widely used in applications where an autonomous agent is influencing its surrounding environment through actions [***](An MDP-based Recommender System). It is defined by a four-tuple $\langle S, A, R, Pr\rangle$ representing a Markov Chain with values and decisions that follow the following Markov property: *a state S is Markov if and only if* $\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, ...S_t]$*. In other words, the future is independent of the past given the present.* If a reinforcement learning task satisfies the Markov property and the state and action spaces are finite, then it can be modeled as a finite Markov Decision Process.

A particular finite MDP is defined by its state and action sets, a reward function R (equation 2.6) that assigns a real value to each state/action pair, and a state-transition function Pr (equation 2.7) which provides the probability of transitioning between every pair of states given each action. Altogether, these quantities completely specify the most important aspects of the dynamics of a finite MDP.

$$\mathcal{R}^a_{ss'} = \mathbb{E}r_{t+1}|s_t = s, a_t = a, s_{t+1} = s' \tag{2.6}$$

$$\mathcal{P}^a_{ss'} = Prs_{t+1} = s'|s_t = s, a_t = a \tag{2.7}$$

In an MDP, the decision-maker?s goal is to find an optimal policy π, such that at each stage of the decision process, the agent needs only to obtain the current state s and execute the action $a = \pi(s)$. Various exact and approximate algorithms have been proposed for estimating the optimal policy $\pi*$, such as policy iteration, value iteration, Monte-Carlo learning, temporal difference, Q-learning, SARSA, etc[***].

2.2.2 A standard model-free Reinforcement Learning Setup

A standard RL setup consists of an agent interacting with an environment E in discrete timesteps. At each timestep t, the agent receives an observation s_t, takes an action a_t and receives a reward r_t. Additionally, the environment E may be stochastic so it can be modeled as an MDP with a state space $\mathcal{S}$, action space $\mathcal{A}$, an initial state distribution $\rho(s_1)$, transition dynamics $\rho(s_{t+1}|s_t, a_t)$, and reward function $r(s_t, a_t)$. On the other hand, the agent?s behavior is defined by a policy π, which maps states to a probability distribution over the actions $\pi : \mathcal{S} \rightarrow P(\mathcal{A})$. Finally, the return from a state is defined as the sum of the discounted future reward $R_t = \sum_{i=t}^{T} \gamma^{(i?t)} r(s_i, a_i)$. As the return depends on the actions chosen and therefore on π, it may be stochastic.

The goal in reinforcement learning is to learn a policy which maximizes the expected return from a start distribution $J = \mathbb{E}_{r_i, s_i \sim E, a_i \sim \pi}[R_1]$. The action-value function is used in many RL algorithms and describes the expected return after taking an action a_t in state s_t and thereafter following policy π:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_{i \geq t}, s_{i>t} \sim E, a_{i>t} \sim \pi}[R_t | s_t, a_t] \tag{2.8}$$

Moreover, many approaches in RL opt to represent the action-value function as a recursive relationship using the Bellman equation:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}[r(s_t, a_t) + \gamma \mathbb{E}_{a_{t+1} \sim \pi}[Q^{\pi}(s_{t+1}, a_{t+1})]] \tag{2.9}$$

Nevertheless, If the target policy is deterministic we can describe it as a function $\mu : \mathcal{S} \leftarrow \mathcal{A}$ and avoid the inner expectation as shown in equation 2.10. As the expectation depends only on the environment, it is possible to learn Q^{μ} off-policy, using transitions which are generated from a different stochastic behavior policy μ.

$$Q^{\mu}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}[r(s_t, a_t) + \gamma Q^{\mu}(s_{t+1}, \mu_{t+1})] \tag{2.10}$$

Q-learning[***](Q-learning. Machine learning), is a commonly used off-policy algorithm which uses a *greedy* policy $\mu(s) = argmax_a Q(s, a)$ in order to estimate the action that gives the maximum reward. As a result, as the algorithm considers function approximators parameterized by θ^Q, it can be used as an optimizer that minimize the loss:

$$L(\theta^Q) = \mathbb{E}_{s_t \sim \rho^\beta, a_t \sim \beta, r_t \sim E}[(Q(s_t, a_t|\theta_Q) - y_t)^2] \tag{2.11}$$

where $y_t = r(s_t, aa_t) + \gamma Q(s_{t+1}, \mu(s_{t+1})|\theta^Q)$. While y_t is dependent on θ^Q, this is tipically ignored.

2.2.3 Exploration/Exploitation Dilemma

One of the challenges that arise in reinforcement learning is the trade-off between exploration and exploitation[***]. To obtain a lot of reward, a reinforcement learning agent must prefer actions that has already tried in the past and found to be effective in terms of reward. But to discover such actions, it needs to try actions that it has not been selected before. In other words, The agent has to exploit actions that are already known as effective, but it also has to explore in order to make better action selections in the future.

Naïve approaches use a greedy policy $\pi(s) = argmax_a Q(s, a)$ to get the action with maximum reward as a pure exploitative model. However, an ϵ-greedy policy can be considered instead to allow exploration in the environment and improve the estimation of the non-greedy action values. Therefore, the agent will select a random action with a probability ϵ, and will act greedily with probability $1 - \epsilon$. The advantage of ϵ-greedy policy over greedy policy are the former continuously explore and improve the chances of recognizing possible actions that can lead to better rewards.

2.2.4 OpenAI Gym: a Reinforcement Learning Toolkit

OpenAI Gym[1] is a toolkit for developing and comparing reinforcement learning algorithms[***](OpenAI Gym). It includes a growing collection of benchmark problems that expose a common interface that abstracts the RL framework described above. It focuses on the episodic setting

[1] `https://gym.openai.com/`

of reinforcement learning, where the agent?s experience is broken down into a series of episodes. During each episode, the agent?s initial state is randomly sampled from a distribution, and the interaction proceeds until the environment reaches a terminal state. The final goal in OpenAI Gym is to maximize the expectation of total reward per episode, and to achieve a high level of performance in as few episodes as possible.

The design of the framework is based on the authors? experience developing and comparing reinforcement learning algorithms. It manages two core concepts: (1) a free-to-code *agent* that maximizes the convenience for users allowing them to implement different styles of agent interface; and (2) a common *environment* interface that ease the implement and testing of different reinforcement learning problems under the same framework. Moreover, environments are strictly versioned in order to maintain the consistency in the results obtained by previously implemented agents; and are instrumented with a configurable monitor interface, which keeps track of every time step during simulation.

In general, OpenAI Gym contains a collection of Partially Observed MDP Environments (POMDPs). After performing an action step in an environment, its interface returns the observation of current state, reward achieved by the action performed, a done flag indicating if an episode ends (and therefore, needs to be reset), and an information object containing useful information for debugging and learning. Furthermore, environments come with first-class space objects that describe the valid actions and observations. Such spaces can be defined as a *discrete* object allowing a fixed range of non-negative numbers, or as an n-dimensional *box space* for representing continuous spaces.

On the other hand, the measure of performance in a RL algorithm under an environment can be measured along two axes: *final performance* or average reward per episode, after learning is complete; and, *sample complexity* or the amount of time it takes to learn. The latter can be measured in multiple ways, for example, by counting the number of episodes until reaching a threshold level of average performance. Naturally, this threshold is chosen per-environment in an ad-hoc way. In addition, the OpenAI Gym website allows users to compare the performance of their algorithms in a way to collaborate with each other rather than a competition.

2.3 Recommender Systems and Reinforcement Learning

Although reinforcement learning seems to have great potential for improving the recommendation problem, it has received relatively little attention and found only limited application. The first experiments on using reinforcement learning techniques were focused on developing recommendation systems for web content and using implicit data from server log files which stored the navigation of the users throughout their contents.

Exploration/Exploitation in Adaptive Recommender Systems considered reinforcement learning with an exploration/exploitation approach of unknown areas to automatically improve their recommendation policy for helping users to navigate through an adaptive web site. They state that a model-free algorithm like Q-Learning can become infeasible if the number of actions are so high. Additionally, they showed that a greedy policy can indeed lead to a suboptimal recommendation model as the use of such kind of policy function, where the b̈estäctions (e.g with higher probability) are taken, does not always improve the policy, so to avoid reinforcing values for states with initial high values, a random exploration process needs to be added to the action selection. The popular exploration ϵ greedy policy algorithm which modifies the action selection by selecting a greedy action with probability $(1-\epsilon)$ or an arbitrary action with probability ϵ otherwise in order to explore other existing options for the user. However, the exploration/exploitation dilemma came out as the exploration makes the future of a sequence less predictable influencing the estimation of the value function itself but inconsistent random actions have to be chosen to obtain a better policy. They finally demonstrated that this problem is solved by starting the algorithm with a lot of exploration and gradually reduce the parameter ϵ when the policy is getting closer to the optimum. Results concluded that a recommender system without exploration potentially can get trapped into a local maxima.

New Recommendation System Using Reinforcement Learning presented a general framework for web recommendation that learns directly from customer's past behavior by applying an RL process which uses the SARSA method[***ref here] and a ϵ-greedy policy. The system was composed of two models: a global model to keep track of customers trends as a whole, and a local model to record the user's individual browsing history. In general, the model treats pages as states of the system and links within a page as actions. To predict the next state, it used a ranking system that is also separated into 2 parts. i) a global ranking system using the data from a Q_{global}-matrix obtained from the action-value function to choose the next state by applying an ϵ-greedy policy. It gave the chance for rank new items that have few clicks but may match a user's interest. ii) a local ranking Q_{local} which uses an inverse ϵ-greedy policy to estimate the next state. It gave to customers more chance to explore other products than the ones they already visited. As a result, the system finds the final total reward using $Q_{total} = Q_{local} + wQ_{global}$ where $w \in (0 - 1]$ is a weight hyper parameter of the model.

Authors performed some experiments to find a relationship between ϵ and the user click rate. The findings showed that a value of $\epsilon = 0.2$ preserves the balance between exploration and exploitation, meanwhile If $\epsilon < 0.2$ the system will gave small chances to users to explore new items, or may include products that do not match to the customer's interest otherwise ($\epsilon > 0.2$). Even if the purpose of Q_{local} was to promote new products, it was less effective than Q_{global}. In conclusion, the result of this experiments explained that if a user is allowed to do much exploration, she has a chance to find non-relevant products, but if users just exploit the system, they have a chance to just keep receiving the same recommended items over time.

An MDP-Based Recommender System argue that it is more appropriate to formulate the problem of generating recommendations as a sequential optimization problem so they described a novel approach for a commercial web site based on Markov Decision Processes (MDPs) together with a novel predictive model that outperformed previous models. An MDP-based recommender system can take into account the expected utility and the long-time effect of a particular recommendation, and suggest an item whose immediate reward is lower, but leads to more *profitable* rewards in the future. Moreover, these considerations are taken into account automatically by any optimal policy generated for an MDP model which ensures a balance in the generation of the maximal expected reward stream. However, the previous benefits are offset by the fact that the model parameters are unknown, randomly initialized and that they take considerable time to converge. So, they introduced in their work a model-based system for collaborative filtering with a strong initial model, which solves quickly, and that does not consume too much memory.

Before implementing the recommender, they initialize a predictive model of user behavior using data extracted from the web site, and then use it to provide the initial parameters for the MDP. The predictive model can be thought of as a first-order Markov chain (MC)[***] of user dynamics in which states correspond to sequences of *k* events (in this case: previous selections) representing relevant information about the users interaction. They experimented with small values of k ($k \in [1-5]$) in order to overcome data sparsity and MDP solution complexity issues. Thus, the transition function describes the probability that a user whose k recent selections were $x_1, ..., x_k$ will select the item x' next. Finally, enhancements on the maximum-likelihood n-gram formulation[***] were proposed in order to find a good estimate of the transition function for the initial predictive model without suffering the problem of data sparsity and bad performance.

The proposed maximum-likelihood estimation include three major enhancements. First, a form of skipping based on the observation that the occurrence of the sequence $x_1, x_{2,x}3$ lends some likelihood to the sequence x_1, x_3. In other words, after initializing the counts of each state transition using the observed data, for any given user sequence $x_1, x_2, ..., x_n$, they added a fractional count $1/2^{(j-(i+3))}$ to the transition from $s = \langle x_i, x_{i+1}, x_{i+2} \rangle$ to $s' = \langle x_{i+1}, x_{i+2}, x_j \rangle$, for all $i + 3 < j?n$, which acts as a diminishing probability of skipping a large number of transactions in the sequence. Equation 2.12 shows the updated formulation of the (normalized)maximum-likelihood estimation, where $count(s, s')$ is the fractional count associated with the transition from s to s'.

$$tr_{MC}^{base}(s, s') = \frac{count(s, s')}{\sum_{s'} count(s, s')} \tag{2.12}$$

The second enhancement exploits the similarity of sequences in a form of clustering. The idea is that the likelihood of transition from s to s' can be predicted by occurrences from t to s', where s and t are similar. Equation 2.13 define the similarity of states s_i and s_j where $\delta(\dot{,})$ is the Kronecker delta function and s_i^m is the m-th item in state s_i. Then, the similarity count from state s to s' was defined using equation 2.14. Finally, the new transition probability from s to s' given by equation 2.15 yielded the best results during evaluation.

$$sim(s_i, s_j) = \sum_{m=1}^{k} \delta(s_i^m, s_j^m) \cdot (m+1) \tag{2.13}$$

$$simcount(s, s') = \sum_{s_i} sim(s, s_i) \cdot tr_{MC}^{base}(s, s') \tag{2.14}$$

$$tr_{MC}(s, s') = \frac{1}{2} tr_{MC}^{base}(s, s') + \frac{1}{2} \frac{simcount(s, s')}{\sum_{s''} simcount(s, s'')} \tag{2.15}$$

Due to larger values of k lead to states that are more informative whereas smaller values of k lead to states with more statistical meaning, they added as a third enhancement a finite mixture modeling (e.g. combine a trigram, a bigram and a unigram into a single model) in a way to balance these conflicting properties by mixing k models, where the *i-th* model looks at the last i transactions. In addition, they applied mixture weights during experiments as $\pi_1 = \ = \pi_k = 1/k$ so the generation of the models for smaller values entails little computational overhead.

An evaluation on the accuracy of the predictive model was performed using real user transactions and browsing paths (from web logs) from the commercial store web site. Besides, they compared different variations of their enhancements on the MC approach with other models like i)both sequential and non-sequential form of the *Microsoft Commerce Server 2000*(Heckerman et al. (2000))[***] (local distributions are probabilistic decision trees); and ii)unordered versions of MC models. Results outlined that the MC models using skipping, clustering, and mixture modeling yielded better results than any other other variation of model *i*. Sequence-sensitive models outperformed the accuracy of non-sequential models, while MC models were superior to the predictor models. and the skipping enhancement was only beneficial for the transactions data set

On the other hand, the MDP-based recommender system, models the recommendation process as a sequence of states and attempts to optimize it. Here, the predictive model played an important role in the construction of the model as it provides the probability, denoted by

$Pr_{pred}(x|x_1, ..., x_k)$, that a user purchase a particular item x given her sequence of past purchases $x_1, ..., x_k$. The components of the reinforcement learning model are defined as follows: a) the states are the k-tuples of items purchased; b) the actions correspond to a recommendation of an item; c) the rewards depends on the last item defining the current state only, where its net profit of the item was used; and d) the transition function is the stochastic element of the model and estimates the user's actual choice. In brief, equation 2.16 represents the transition function or the probability that a user will select item x'' given that item x' is recommended in state $\langle x_1, x_2, x_3\rangle$.

$$tr^1_{MDP}(\langle x_1, x_2, x_3\rangle, x', \langle x_2, x_3, x''\rangle) \tag{2.16}$$

In order to solve the MDP, the policy iteration algorithm was used as their state space enjoys of the following features that lead to fast convergence: i) the inherent directionality that transitions showed are useful to reduce the running time of the solution algorithm; ii) the computation of an optimal policy is not sensitive to variations in the number of k past transactions a state represent; iii) ignoring unobserved states by maintaining transition probabilities only for states where a transition actually occurred; and iv) using the independence of recommendations or Markov property, where the probability that a user buys a particular item depends only on her current state, allowing the algorithm to handle large action spaces.Tests performed on real data showed that the policy iteration converges after a few iterations.

To update the model, the system used an off-line approach which keeps track of the recommendations and the user selections and build a new model at fixed time intervals (e.g. once a week). So, to re-estimate the transition function at time $t + 1$ the following counts are obtained:

$$c^{t+1}_{in}(s, r, s \cdot r) = c^{t}_{in}(s, r, s \cdot r) + count(s, r, s \cdot r) \tag{2.17}$$

$$c_{out}^{t+1}(s,r,s\cdot r) = c_{out}^{t}(s,r,s\cdot r) + count(s,s\cdot r) - count(s,r,s\cdot r) \tag{2.18}$$

$$c_{total}^{t+1}(s,s\cdot r) = c_{total}^{t}(s,r,s\cdot r) + count(s,s\cdot r) \tag{2.19}$$

$$tr(s,r\in R,s\cdot r) = \frac{c_{in}^{t+1}(s,r,s\cdot r)}{c_{total}^{t+1}(s,s\cdot r)} \tag{2.20}$$

$$tr(s,r\notin R,s\cdot r) = \frac{c_{out}^{t+1}(s,r,s\cdot r)}{c_{total}^{t+1}(s,s\cdot r)} \tag{2.21}$$

$$c_{in}^{0}(s,r,s\cdot r) = \xi_S \cdot tr(s,r,s\cdot r) \tag{2.22}$$

$$c_{out}^{0}(s,r,s\cdot r) = \xi_S \cdot tr(s,r,s\cdot r) \tag{2.23}$$

$$c_{total}^{0}(s,s\cdot r) = \xi_S \tag{2.24}$$

where Equation 2.17 corresponds to the number of times a recommendation r was accepted in state s, equation 2.18 is the number of times a user select item r in state s even though it was not recommended, equation 2.19 is the number of times a user pick item r while being in state s, regardless of whether it was recommended or not, and finally, equations 2.20 and 2.21 represent the actual probability that a user will select item r given state s when r was recommended or not respectively.

The transition function had to be carefully initialized in order to be fairly accurate when the system was first deployed to avoid the cold-start problem, so the at time $t = 0$ were calculated using equations 2.22 2.23 2.24, where $\xi_s = 10 \cdot count(s)$, causing states that are infrequently observed to be updated faster than already observed states. Moreover, when the first transition to a state $s \cdot r$ is observed, its probability is initialized to 0.9 the probability of the most likely next item in state s with $\xi_s = 10$. In this way, the system balances the need to explore unobserved items in order to improve its model by recommending non-optimal items occasionally until getting their actual counts.

Finally, an empirical validation of their thesis compared the performance in terms of their value or utility and computational cost of the MDP-based recommender system with the performance of a recommender based on the predictive model (MC) as well as other variants. The deployed system was built using three mixture components (combined using an equal weight), with history length $k \in [1, 3]$ for both the MDP and MC model, and used the policy-iteration procedure and approximations. The average profit generated by users using the site was 28% higher for the MDP group. while in terms of computational costs, the MDP-based model provided fastest recommendations at the price of more memory use, and also built models more quickly.

All in all, the approach presented make use of off-line data to initialize a model in order to provide an adequate initial performance and overcome the cold-start problem, the authors avoid using some form of reinforcement learning technique as they argue that its implementation requires many calls and computations by a recommender system online, which will lead to slower responses, producing undesirable results for the web site owner.

Usage-Based Web Recommendations: A Reinforcement Learning Approach proposed a machine learning perspective based on a off-line reinforcement learning model to solve the recommendation problem using the Q-Learning algorithm while employing concepts and techniques commonly applied in the web usage mining domain. They argued that their decision on using this method seems appropriate for the nature of web page recommendation problem as it provides a mechanism which is constantly learning, does not need periodic updates, can be easily adapted to changes in the website structure and new trends in users behavior. Consequently, the system was trained using web usage logs available as the training set and experimental evaluations were carried out in order to demonstrate how this approach can improve the quality of web recommendations.

The RL problem formulation was considered as as a competition between different recommender systems to gather more points, than a 2-

player game, as in the recommender system environment, self-play, a typical technique used in training RL systems[***], cannot be used to train the system so the system needed of actual web usage data for training. Moreover, the model, which has a stochastic nature, consists of the following properties:

states: showing the history of pages visited by the user so far. For this case, a notion of N-Grams was adopted and a sliding window of size w was set to limit the page visit sequences to a constant number and avoid large state spaces.

actions: consists of a single page recommendation at each state.

policy: computes the reward of performing an action a in state s. It rewards actions positively if it recommends a page that will be visited in one of the consequent states (not necessarily the immediate next state). Formally, rewards are dependent on the next state, the intersection of previously recommended pages in each state and the current page sequence of the state.

As the model does not have a predetermined reward function $R(s,a)$ or a transition function $\delta(s,a)$ to give the next state, the reward can be estimated by considering each state s is formed by two sequences $V_s = \langle p^V_{s,1}, p^V_{s,2}, ..., p^V_{s,w} \rangle$, and $R_s = \langle p^R_{s,1}, p^R_{s,2}, ..., p^R_{s,n} \rangle$, indicating the sequence of visited and previously recommended pages respectively, where $p^V_{s,i}$, indicates the ith visited page in the state and $p^R_{s,i}$ indicates the ith recommended page in the state s, Additionally, they had two considerations in the implementation of the reward function: i) only the occurrence of the last page visited in the recommended pages list in state s' is used to reward the action performed in the previous sate s, and ii) the time the user spends on a page, assuming that the more time a user spends on a page the more interested. So, the reward function was defined as follows:

1. Assume $\delta(s,a) = s'$

2. $P_R = V_{s',w} \cap R_{s'}$
3. if $p \neq \emptyset$
4. For page p in P_R
   (a) $r(s,a) += reward(Dist(Rs',p), Time(p_w^v))$

where $Dist(R_i, p)$ is the distance of page p from the end of the recommended pages list and $Time(p_w^v)$ indicates the time user has spent on the last page of the state. Finally, The $reward(Dist, Time)$ function was defined as a linear combination of both values as $reward(Dist, Time) = \alpha \times dist + \beta \times Time$ with $\alpha + \beta = 1$.

Subsequently, the definition of the learning algorithm also took into consideration the following characteristics: 1) the Q-Learning algorithm given by equation 2.25 was proposed as an update rule and the structure to estimate how successful a prediction can be. Here, the decreasing value of α_n caused these values to gradually converge and decreases the impact of changing reward values as the training continues; 2) an ϵ-greedy action selection was picked as it is important to add some exploration of the state space especially at the beginning of the training; 3) the algorithm followed a TD(0) off-policy learning[***] procedure, as the maximum Q value of the next state is considered when estimating the future reward; and 4) the computation of $r(s,a)$ suffered a slightly change by considering value of $Q(s',a)$ with a coefficient γ in order to propagate the value of performing a specific action beyond the limits imposed by the w. As the authors mentioned in their work: "the basic idea is that when an action/recommendation is appropriate in state Si, indicating the recommended page is likely to occur in the following states, it should also be considered appropriate in state S_{i-1} and the actions in that state that frequently lead to S_i". During the training phase, the described algorithm converged after a few thousand (between 3000 and 5000) visits of each episode or user session.

$$Q_n(s,a) = [(1-\alpha_n)Q_{n-1}(s,a) + \alpha_n[r(s,a) + \gamma \max_{a'} Q_{n-1}(\delta(s,a),a')] \quad (2.25)$$

with

$$\alpha_n = \frac{1}{1 + visits_n(s,a)} \quad (2.26)$$

For the purpose of performing a set of experimental evaluations of the proposed model, simulated log files generated by a web traffic simulator were used to train and tune the rewarding functions. The metric used for each evaluation were Recommendation Accuracy, Coverage (similar to precision and recall metrics) and Shortcut Gain (measures how many page-visits users can save if they follow the recommendations). First of all, an experiment with different values of w showed that fixed window size of 3 as recommendation history resulted on better accuracy and shortcut gain, so this value was set for the rest of system evaluations. Next, they experimented the impact of gradually increasing (in steps of 5%) the coefficient α of parameter *Dist* in the reward function, resulting on both higher accuracy and higher shortcut gain for values up to 15%. This matched the natural consequence of adding a bounded size of window on recommendations history. Finally, a set of experiments also tested the system performance by increasing the coefficient γ in the reward function, obtaining an increase of the accuracy until reaching an upper bound (around 0.20%) where it began to drop, while the shortcut gain increased steadily up to a point where recommendations became so inaccurate.

Briefly, the proposed off-line reinforcement learning method used a simple formulation of the recommendation problem but it obtained much better results compared with two baselines methods: association rules and item-based collaborative filtering (with probabilistic similarity measure). As the coverage increased, naturally accuracy decreased in all systems, but the RL approach outperformed the other two systems displaying a lower rate in which its accuracy decreased. Authors concluded that the algorithm is a good candidate for solving the recommendation problem as it does not rely on any previous assumptions regarding the probability distribution of visiting a page after having visited a sequence of pages, and that the nature of the problem matches perfectly with the notion of delayed reward (known as temporal difference). In addition, they suggested that it could be possible to use a more complicated formulation of the reward function rather than a linear combination of factors, such as a neural networks, in order to produce better results.

Later, in **A Hybrid Web Recommender System Based on Q-Learning** they exploited the previous reinforcement learning framework and present a hybrid web recommendation method enriched with semantic knowledge about the usage behavior as a way to improve the RL solution and obtain a more generalized solution regarding to the usage data it has. The new system used the incremental DCC[***] clustering algorithm to map pages to higher level concepts, and thus exploit the hierarchical and conceptual document clustering to provide a semantic relationship between them in combination with the usage data in a user session. Therefore, each state now consists of a sequence of concepts visited by the user, while actions are recommendation of pages that belong to a specific concept. This definition resulted in a much smaller state-action space as the size of the state space is now dependent on the number of distinct page clusters.

Now, an action a recommending a concept c is rewarded if the user visits a page belonging to concept c later in his browsing session. The new reward function shown in equation 2.27 takes into account the content similarity of the recommended and visited pages, where CBR represents the content-based reward of an action (which is equal to the similarity score between concepts) defined in equation **??**, and UBR is the usage-based reward defined in the previous approach.

$$UBR(Dist(R_{s'}, r), Time(P_{t+1})) \times CBR(r, C(P_{t+1})) \tag{2.27}$$

$$l(c) = -log_{\surd}(c) \tag{2.28}$$

$$Sim(c_1, c_2) = max_{a \in LCA} l(a) \tag{2.29}$$

To sum up, the algorithm did not make predictions based on weak usage patterns as now the states represent a generalized view of many single visit sequences. However, this approach seems specifically appropriate for the off-line training phase. On the other hand, evaluation results showed the flexibility of the new RL approach to incorporate different sources of information to improve the quality of recommendations, and also demonstrated how it could be extended in order to incorporate various sources of information.

Improving adaptation of ubiquitous recommender systems by using reinforcement learning and collaborative filtering proposed a learning system for Context-based Recommender System (CBRS)[***] by modeling an MDP agent which combines a hybrid Q-learning algorithm (HQL), collaborative filtering and case-based reasoning techniques in order to define a *contextual* recommendation process based on different context dimensions (cognitive, social, temporal, geographic). It addressed the following problems that came out in recommender systems: a)avoid the intervention of experts as the use of the Q-learning algorithm does not need initial user?s information; b)reduce the cold start problem thanks to the ability of the Q-learning algorithm to explore the knowledge of other users in the same context by using CF; c)accelerate the learning process by mixing Q-learning with case-based reasoning techniques (reuse of cases yields to faster user satisfaction); and d)an exploration strategy allows recommendations to adapt to the user's interest evolution.

As a result, the model was based in three main algorithms. First, a hybrid-based CF approach which combines the advantages of the memory-based (fill the missing rating values of the user-item matrix) and model-based CF (form the nearest neighbors of each item). Second, the Case based reasoning (CBR)[***] algorithm was picked as it uses knowledge of previous cases to solve new problems, by finding a similar past case and thus reusing it to solve the current situation. Finally, the Q-learning algorithm was improved by: i)reusing past cases information gathered from the CBR, and ii)giving the ability to use information from other users sharing the same interests, by extending the ϵ-greedy strategy to select a random action based on the similarity of user profiles (obtained from the CF algorithm).

The global mechanism of a context-based recommender system (CBRS) which uses the HyQL algorithm was composed mainly by the following modules:

sensing module: detects time, location, cognitive and social dimensions of the context.

thinking module: composed by an abstraction phase that is based on inference rules defined on temporal and space ontologies, and an aggregation phase the two dimensions.

reasoning module: chooses an action based on the HyQL algorithm

Finally, evaluations comparing the Q-learning and HyQL with respect to solving the cold start problem were performed, and consisted on testing the precision of the first 100 trials of the system starting when the user is connected to the system. In General, results showed that the precision of HyQL was greater than the precision of Q-learning, demonstrating that this approach is a good candidate to be used in the recommender system problem. However, as the system implementation depends on the context data, it needs of a hand-crafted feature creation process in order to be used on different situations.

Exploration in Interactive Personalized Music Recommendation: A Reinforcement Learning Approach formulated an interactive and personalized music recommendation approach as a reinforcement learning task based on the multi-armed bandit problem, where songs were treated as arms and user ratings as payoffs, and the objective function was set to maximize the cumulative reward of a targeted user over the long term. The model considered the exploration-exploitation trade-off and a Bayesian model in order to deal with both audio content and the novelty of recommendations and thus learn the user musical preferences online based on their feedback.

The model differs from other approaches in three aspects: i)it is based on audio content; ii)the model is highly efficient as it allows easy online updates; and iii)the model is evaluated based on online real-life user interaction data and does not need a test dataset. Moreover, each recommendation serves two objectives: (1) satisfy the user?s current musical taste, and (2) obtain user feedback needed to improve future recommendations. Even though authors mentioned that an MDP formulation can model a broader range of problems than the multi-armed bandit, it requires much more data to train and is often more computationally expensive.

Their choice to work with a the content-based approach rather CF was justified for the following reasons. First, due to the bayesian nature of their formulation, methods for matrix factorization (MF) are much more complicated than the proposed linear model. Second, MF often needs of large amount of data for training. Third, existing Bayesian MF methods are inefficient for online updating, so they do not fit in a bandit model which is updated once a new rating is obtained, Fourth, a content-based method does not suffer of the new song problem. Finally, content-based approaches capture causality within music content rather than pure correlation that CF methods offer.

Subsequently, the interactive, personalized recommender system used a reinforcement learning task called the multi-armed bandit[***] which addressed both the exploration-exploitation trade-off and playlist generation with a single unified model. Whereas some RL models considered only a greedy strategy that does not actively seek fort user feedback, resulting in suboptimal recommendations over the long term, this model takes into account the uncertainty of both the mean and the variance of the rating distribution, allowing the recommender to explore user preferences actively rather than merely exploiting the available rating information. An approach better than the typical ϵ-greedy method to solve of the multi-armed bandit problem, is the use of the Upper Confidence Bound (UCB)[***] algorithm, but it requires an explicit form of the confidence bound that is difficult to derive in the case of music recommendation.

The personalized music rating model was focused on audio content and novelty. The former considers the overall preference of a song and is represented by the linear function $U_c = \theta'\mathrm{x}$ where x is the feature vector of the audio content and θ is the user preference in different music features (it was considered constant over time). On the other hand, the novelty factor was defined after examining the song's repetition distribution of 1000 users? listening histories collected from the Last.fm dataset. Results showed that most of the songs a user listens to are repeats and that the frequency distribution approximately follows the Zipf?s law[***], where only a small set of songs are repeated most of the time. As a consequence, it was assumed that the novelty of a song decays immediately after it is listened to and then gradually recovers according to the function $U_n = 1 - e^{-t/s}$, where s is a recovery speed (unknown) parameter learned from user interactions and $e^{-t/s}$ is the well-established forgetting curve[***] that measures user's memory retention of a song. In other words, novel songs are those of which that at a certain time a user has little or no memory. It is important to emphasize that the definition of this factor could be different or not applicable to other recommendation contexts than the musical environment.

Overall, the rating model, defined in equation 2.30, assumed that a rating is formed as the combination of the user?s preference of the song?s content and the dynamically changing novelty. Hence, each user was represented by a set of parameters $\Omega = \{\theta, s\}$, where Ω needs to be estimated from historical data, so uncertainty had to be taken into account.

$$U = U_c U_n = \theta' \mathbf{x}(1 - e^{-t/s}) \tag{2.30}$$

With this in mind, the new problem formulation was solved using the Bayesian Upper Confidence Bound (Bayes-UCB)[***] algorithm. In The Bayes-UCB, the true expected payoff U_i defined in equation 2.31 for arm i is treated as a random variable and the posterior distribution $p(U_i|\mathcal{D})$ of U_i given the history of payoffs $\mathcal{D}_\updownarrow = \{(\mathbf{x_i}, t_i, r_i)\}_{i=1}^{l}$ is predicted using equations2.32 and 2.33. Finally, the Bayes-UCB recommends song k^* that maximizes the quantile function $argmax_{k=1...|S|} Q(\alpha, P(U_k|D_l))$ where Q satisfies$\mathcal{P}[Uk \leq Q(\alpha, P(U_k|D_l))] = \alpha$ and $\alpha = 1 - \frac{1}{l+1}$

$$\mathcal{E}[R_i] = U_i = \theta' \mathbf{x_i}(1 - e^{-t_i/s}) \tag{2.31}$$

$$p(\mathbf{\Omega}|D_l) \propto p(\mathbf{\Omega})p(\mathbf{\Omega}|D_l) \tag{2.32}$$

$$p(U_k|D_l) = \int p(U_k|\mathbf{\Omega})p(\mathbf{\Omega}|D_l)\mathbf{d\Omega} \tag{2.33}$$

Since equation 2.32 does not have a closed-form solution, a Markov Chain Monte Carlo (MCMC)[***] should be used as an approximate inference algorithm. However, authors argued that its performance is very low and users could wait for up to a minute until the MC converges. Hence, they developed a Bayesian model using a piecewise-linear function approximation, as well as a variational inference algorithm to approximate posterior distribution of Ω. For more details about the model definition, please refer to section 4.2 in [***](this paper). Authors mentioned that even if the model just considered audio content and novelty of music on its definition, other factors like diversity, mood or genre could be also approximated by linear functions and then added to it. All in all, the defined approximate Bayesian model was used to obtain the fixed-level quantile of $p(U_i|\mathcal{D})$ and thus estimate the expected payoff and confidence bound of the conceptual Bayes-UCB.

Evaluations of both efficiency and effectiveness were carried out over six recommendation algorithms and models (Random, LinUCB-C (Content-based), LinUCB-CN (Content and novelty based), Bayes-UCB-CN, Bayes-UCB-CN-V (Variational Inference), and Greedy-CN) demonstrating that the proposed approaches were accurate and highly efficient. The effectiveness study used the *regret*[***] metric to compare the algorithms, showing that the Bayes-UCB-based algorithms performed better than Greedy-CN due to the balance of exploration and exploitation it offers, whereas the good performance of the Bayes-UCB- CN-V indicated that the piecewise-linear approximation and variational inference were appropriately defined. Moreover, results in the efficiency study empirically proved that the developed variational inference algorithm was 100 times faster than the MCMC. Additionally, a user study and an overall evaluation of recommendation performance dropped good conclusions. Results showed that the bandit approach with the novelty factor addressed the cold-start problem improving the recommendation performance.

To sum up, this work was considered by the authors as the first to balance exploration and exploitation based on reinforcement learning and the multi-armed bandit, and thus improve recommendation performance by addressing the cold-start problem in music recommendation without relying on additional (contextual information). An approximation to the rating model and the new probabilistic inference algorithms helped to achieve real-time recommendation performance that could be generalized to other recommenders and/or media types. Finally, authors suggested that this work could be extended to model the correlations between different users to further reduce the amount of exploration by using hierarchical Bayesian models. On the other hand, if people prefer to boost the performance of CF, they suggested to exploit the exploration/exploitation trade-off idea by using the latent features learned during the matrix factorization rather than the audio features and keep other parts of the proposed system unchanged.

ENHANCING COLLABORATIVE FILTERING MUSIC RECOMMENDATION BY BALANCING EXPLORATION AND EXPLOITATION extended the work presented above by introducing exploration into the CF context. The approach used a Bayesian graphical model that takes into account the CF latent factors and novelty of recommendation, as well as a Bayesian inference algorithm to efficiently estimate the posterior rating distributions.

Authors argued that their previous work, based on a content-based approach, had experienced the following drawbacks: (i) the personalized user rating model suffer of a semantic meaning between low-level audio features and high-level user preferences; (ii) it is difficult to determine which acoustic features are actually effective in the music recommendation scenario; and (iii) recommendation of songs under this approach lacks of variety due to most of them are acoustically similar. Therefore, they presented a memory-based CF approach using matrix factorization in order to improve recommendations.

Recalling the definition of Matrix factorization in Section 2.1.1, this method characterizes users and songs by vectors of latent factors $\mathbf{u_i}$ and $\mathbf{v_j}$ respectively with $i \in [1, m], j \in [1, n]$. In order to learn the latent feature vectors, the system used equation 2.34 and Alternating Least Squares (ALS)[***] to minimize the regularized cost function on the training set:

$$\sum_{(i,j)\in I}(r_{ij}-\mathbf{u_i^T}\mathbf{v_j})^2+\lambda(\sum_{i=1}^{m}n_{ui}\|\mathbf{u_i}\|^2+\sum_{j=1}^{n}n_{vi}\|\mathbf{v_j}\|^2) \tag{2.34}$$

where I is the index set of all known ratings, λ a regularization parameter, n_{ui} the number of ratings by user i, and n_{vj} the number of ratings of song j. However, the traditional CF approach often fails to take into consideration novelty and works greedily.

As a result, a reinforcement learning approach for CF-based music recommendation based on the n-arm bandit problem was proposed. The user rating model considered that song?s rating is affected by two factors: *CF score*, (how much a user likes the song in terms of each CF latent factor) denoted by U_{CF} and defined inn terms of matrix factorization, and *novelty score* (the dynamically changing novelty of the song) denoted by U_N. Thus, the final user rating model, given in equation 2.35, can be defined as a combination of the two scores, where vector θ indicates the user?s preferences for different CF latent factors, $\mathbf{v}$ is the song feature vector learned by the ALS CF algorithm, t is the time elapsed since when the song was last heard, s the relative strength of the user?s memory, and $e^{?t/s}$ the well-known forgetting curve.

$$U=U_{CF}U_N=(\theta^{\mathbf{T}}\mathbf{v})(1-e^{-t/s}) \tag{2.35}$$

In the same way as in [***](previous paper), each user was associated with a pair of parameters $\Omega=(\theta,s)$ to be learned from the user's rating history, and their underlying ratings were considered random rather than fixed numbers (estimated using $\mathcal{E}[R_j]=U_j$). Exploration was also introduced by using both a similar Bayesian Upper Confidence Bound (UCB) sampling algorithm and Bayesian Graphical model to estimate the posterior distribution $p(U_j|\mathcal{D})$ of U_j given the target user?s rating history $\mathcal{D}$, but in terms of the CF latent factors rather than audio content. Then, the song with the highest fixed-level *quantile* value (equation **??**) of $p(U_j|\mathcal{D})$ will be recommended to the target user.

However, the efficiency of the convergence in the Bayesian inference of this approach was improved by developing a specific Gibbs sampling algorithm[***] as in this case it is simple to sample the ratings from the conditional distribution of θ, while, a Metropolis-Hastings (MH) algorithm[***] was used to draw samples of s. For a more detail explanation of the algorithms, please refer to sections 3.2 and 3.3 in [***](this paper).

Efficiency experiments were carried out over to compare the developed sampling implementation and an MCMC algorithm developed in JAGS[2]. The results showed that their proposed Gibbs sampling algorithm is hundreds of times faster than MCMC evidencing the suitability of the algorithm for online recommender systems. Additionally, an online user study which compared the effectiveness of the proposed Bayes-UCB-CF algorithm against the traditional greedy method and the previous implementation Bayes-UCB-Content[***](previous paper), proved that the cumulative average rating of new algorithm significantly outperformed the baselines. All in all, the Bayes-UCB-CF algorithm achieved a better balanced exploration/exploitation trade-off and significantly showed an improvement on its recommendation performance.

In conclusion, this first attempt to remedy the greedy nature of CF approaches could enhance the performance of CF-based music recommendation significantly. Moreover, the reinforcement learning model seems to be applicable to other contexts different from the music environment, as well as it could be deployed together with the content-based RL model from [***](previous paper) in order to build a hybrid framework which combines the strengths of both approaches.

Generating Music Playlists with Hierarchical Clustering and Q-Learning

[2]`http://mcmc-jags.sourceforge.net/`

DJ-MC: A Reinforcement-Learning Agent for Music Playlist Recommendation considered a music recommendation framework for recommending song sequences using a reinforcement learning approach which models the preferences of both songs and song transitions as a MDP. The model-free agent demonstrated to be potentially effective in the domain and particularly good to solve the cold-start problem as it is able to generate personalized song sequences within a single listening session of 25-50 songs and with no prior knowledge of the new user's preferences.

Assuming a finite set of n musical tracks $\mathcal{M} = \{a_1, a_2, ..., a_n\}$ and playlists of length k, the adaptive playlist generation problem is defined as an episodic MDP $\langle S, A, P, R, T \rangle$ with the following components:

- a state space S composed by the ordered sequence of songs played, $S = \{(a_1, a_2, ..., a_i) | 1 \leq i \leq k; \forall j \leq i, a_j \in \mathcal{M}\}$
- an actions space A formed by all possible candidates for being the next song to play, $a_k \in A$ which means $A = \mathcal{M}$
- a deterministic transition function $P(s, a) = s'$ which indicates the existing transitions from state s to s' by taking action a
- an utiliy function R(s, a) derived from hearing song a when in state s
- $T = \{(a_1, a_2, ..., m_k)\}$: the set of playlists of length k.

In order to find an optimal policy $\pi*$ which obtains the most pleasing sequence of songs to the listener, a model-based approach was chosen arguing that even though model-free approaches learn the value of taking an action a from state s directly, it requires a lot of data to converge, and it is considerably scarce in the domain music recommendation (as well as in other kind of applications). On the other hand, model-based formulations often requires a lot of computation to find an approximate solution to the MDP, but the trade-off of computational expense for data efficiency makes the model-based approach a good option for this problem.

Therefore, since P is deterministic, only the listener?s reward function R required to be modeled. It was compactly represented as the sum of two distinct components: 1) the listener?s preference over songs, $R_s : A \to \mathcal{R}$ and 2) her preference over transitions from songs played to a new song, $R_t : S \times A \to \mathcal{R}$. Hence, R was defined using equation 2.36.

$$R(s,a) = R_s(a) + R_t(s,a) \tag{2.36}$$

In essence, the model represent each song as a vector of spectral auditory descriptors, however, the framework is in principle robust and agnostic to the choice of a specific song corpus. First, the listener reward (linear) function over songs R_s generates a binary feature vector using a sparse encoding of the song descriptors as $R_s(a) = \phi_s(u) \cdot \theta_s(a)$, where $\phi_s(u)$ represents the listener pleasure for a particular set of active features. Then, the listener reward (linear) function over transitions R_t obtains a sparse binary feature vector $R_t(a_i, a_j) = \phi_t(u) \cdot \theta_t(a_i, a_j)$, where $\phi_t(u)$ is a user-dependent weight vector and θ_t is a binary feature vector, with both representing transitions between 10-percentile bins of the song descriptors they share (for learnability purposes). An evaluation on the expressiveness of feature representation showed that different sequences are indeed distinguishable. As a result, the compact representation of songs was still rich enough to capture meaningful differences in user’s preferences, and the system was able to leverage knowledge using a few transition examples to plan a future sequence of songs.

On the other hand, the agent architecture is composed by two submodules: a module for learning the listener parameters (ϕ_s and ϕ_t) which performs the initialization and learning on the fly processes; and a module for planning a sequence of songs and thus estimate the next appropriate song to play. The initialization is divided in two parts: 1) the initialization of the song preferences polls the listener for her k_s favorite songs in the database and then update the value $\phi_s(u)$ using the feature descriptors of each of the selected items; and 2) the initialization of the transition preferences based on presenting different possible transitions that encapsulate the variety in the dataset, and directly asking which of a possible set of options the listener would prefer. Updates to the user feature vector $\phi_t(u)$ are carried out in the same manner as the initialization of song preferences.

After initialization, the agent begins playing songs for the listener, waiting for her feedback, and updating ϕ_s and ϕ_t accordingly. The latter is computed by using a single unified reward signal that considers the relative contributions of the song and transition rewards to set weights (equations 2.37 and 2.38) for credit assignment (equations 2.39 and 2.40). In general, the learning of the fly procedure is perceived as a temporal-difference update with an attenuating learning rate that balances the trust between the previous history of observations and the newly obtained signal.

$$w_s = \frac{R_s(a_i)}{R_s(a_i) + R_t a_{i-1}, a_i} \tag{2.37}$$

$$w_t = \frac{R_t(a_{i-1}, a_i)}{R_s(a_i) + R_t(a_{i-1}, a_i)} \tag{2.38}$$

$$\phi_s = \frac{i}{i+1} \cdot \phi_s + \frac{i}{i+1} \cdot \theta_s \cdot w_s \cdot r_{incr} \tag{2.39}$$

$$\phi_t = \frac{i}{i+1} \cdot \phi_t + \frac{i}{i+1} \cdot \theta_t \cdot w_t \cdot r_{incr} \tag{2.40}$$

After determining the MDP reward function, a Monte Carlo Tree Search[***] (MCTS) heuristic is used for planning. The process chooses a subset of 50 % of the songs in the database with highest R_s score, and then at each point, it simulates a trajectory of future songs selected at random (instead of asking the user her top-k songs) and calculate the expected payoff of the song trajectory using R_s and R_t. This iterative process ends when it finds the trajectory which yields to the highest expected payoff. Finally, the first item of this trajectory is selected to be the next song.

However, a re-planning must run at every step as the system actively adjusts ϕ_s and ϕ_t online based on user feedback. It could arise a problem of high complexity when the song space is very large. Therefore, to mitigate the problem, the agent exploits the structure of the song space by clustering songs according to song types. The solution is implemented using the canonical k-means algorithm (but any clustering algorithm could work) and results on a radical reduction of search complexity.

An evaluation of the cumulative reward distribution of the proposed approach against two baselines (a random agent and a greedy agent) showed that overall, the DJ-MC agent outperforms both random and greedy agents, while in terms of adding the transition reward preferences into the model, the new system presented a small but significant boost in performance compared to a model based only on reasoning about song preferences. All in all, this approach demonstrated to be a good improvement as a reinforcement learning model for music recommendation, as it enables learning from relatively few examples, and the representation captures enough of real peoples? transition reward to provide quality of recommendation sequences.

Hybrid Collaborative Filtering with Neural Networks

2.4 Deep Reinforcement Learning

Notably, recent advances in deep neural networks , in which several layers of nodes are used to build up progressively more abstract representations of the data, have made it possible for artificial neural networks to learn concepts such as object categories directly from raw sensory data

Tousereinforcementlearningsuccessfully insituations approaching real-worldcomplexity,however, agents are confronted witha difficult task: theymust derive efficient representations of the environment from high-dimensional sensory inputs, and use these togeneralizepast experience tonewsituations.

While reinforcement learning agents have achieved some successes in a variety of domain, their applicability has previously been limited to domains in which useful features can be handcrafted, or to domains with fully observed, low-dimensional state spaces.

Playing atari with deep reinforcement learning and **Human-level control through deep reinforcement learning** presented the first deep learning approach model which uses an end-to-end reinforcement learning approach to learn control policies directly from a high-dimensional raw input space. The model, applied to a range of Atari 2600 games, consists of convolutional neural network and an experience replay mechanism[***] to alleviate the problems of correlated data and non-stationary distributions in typical RL problems. Moreover, the model's architecture uses only the state representation as input, and a separate output unit for each possible action (target network), corresponding to the predicted Q-values of each individual action for the input state. This configuration allows the computation of for all possible actions reward in a given state with only a single forward pass. Finally, the resulting neural network is trained with a variant of Q-learning which learns from raw pixels input and the underlying environment properties, and outputs a value function estimating the future rewards.

The formulation models the πenvironment $\mathcal{E}$ as a Markov Decision Process composed by a state space $\mathcal{S}$, an action space $\mathcal{A} = \{1...K\}$, a scalar reward function $r(s_t, a_t)$ and a transition dynamics $\rho(s_{t+1}|s_t, a_t)$. Moreover, the algorithm follows the basic idea behind many reinforcement learning tasks: given the agent's behavior defined by a policy π, they estimate the action-value function by using an approximator function of the Bellman equation (presented in eq. 2.9) as an iterative update. A Q-network function approximator with weights θ is then trained by minimising the sequence of loss functions $L_i(\theta_i)$ defined in equation 2.41

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}[(y_i - Q(s,a;\theta_i))^2] \tag{2.41}$$

where $y_i = \mathbb{E}_{s'\sim\mathcal{E}}[r + \gamma max_{a'} Q(s',a';\theta_{i-1})|s,a]$ is the target for iteration i and $\rho(s,a)$ is a probability distribution over sequences of states s and actions a known as behavior.

Rather than computing the full expectations in the gradient of the loss function, they considered stochastic gradient mini-batch updates of the weight parameters, and a learning algorithm which is: a)*model-free*: replaces the expectations by directly using uniform samples from the behaviour distribution ρ, and b)*off-policy*: it follows an ϵ-greedy strategy of the behaviour distribution that ensures an adequate exploration of the state space.

Evaluation results of the proposed algorithm showed that this method is able to learn how the value function evolves for a reasonably complex sequence of events. Furthermore, a performance evaluation compared this approach with the Sarsa[***] and Contingency[***] methods demonstrating that the deep reinforcement learning algorithm outperforms other methods which need to incorporate significant (handcrafted) prior knowledge about the visual problem to get considerable performance. Finally, the proposed method also achieved better performance than an expert human player in 29 out of 49 games.

The main advantages of using this deep learning approach with an experience replay of size N are: i) each step of experience is potentially used in many weight updates, allowing greater data efficiency; ii) randomizing the samples breaks the correlation in a sequential event and therefore reduces the variance of the updates; iii) the experience replay mechanism allows the behavior distribution to be averaged over many of its previous states, smoothing out learning and avoiding instability or divergence in the parameters; and iv) the method showed that can be applied to a series of different reinforcement learning tasks with no adjustment of its architecture or learning algorithm.

However, this model is only able to handle discrete low-dimensional action spaces and cannot be directly applied to tasks under the continuous domain as the iterative process of finding the action that maximizes the action-value function will become intractable. Additionally, a naive discretization of the action space is not a good solution as it will throw away information about the structure of the action domain.

Continuous control with deep reinforcement learning adapted the ideas presented above to the continuous space and action domain and provided an actor-critic, model-free algorithm based on the Deterministic Policy Gradient[***] (**Deterministic Policy Gradient Algorithms**) that is able to find policies end-to-end whose performance is competitive compared to previous approaches based on a planning algorithm with access to the dynamics of the domain. To do so, they used the similar features of DQN (network architecture, experience replay memory, target outputs, etc), along with batch normalization[***], to overcome the internal covariate shift problem by normalizing the network layer inputs during mini-batches.

The Deep Deterministic Policy Gradient (DDPG) implementation uses an actor-critic approach based on the DPG algorithm which mainly maintains two functions: 1)the actor function $\mu(s|\theta_\mu)$ that obtains the current policy by deterministically mapping states to specific actions; and 2)the critic function $Q(s,a)$ learned by applying the Bellman equation. DPG then updates the actor by applying the chain rule to the expected return from a start distribution $J = \mathbb{E}_{r_i, si \sim E, a_i \sim \pi}[R_1]$ and with respect to the actor parameters:

$$\begin{aligned} \nabla_{\theta^\mu} J &\approx \mathbb{E}_{s_t \sim \rho^\beta}[\nabla_{\theta^\mu} Q(s, a|\theta_Q)|_{s=s_t, a=\mu(s_t|\theta_\mu)}] \\ &= \mathbb{E}_{s_t \sim \rho^\beta}[\nabla_a Q(s, a|\theta_Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s=s_t}] \end{aligned} \tag{2.42}$$

Even if convergence is no longer guaranteed due to the non-linearity of the proposed approximation function, it produces a good generalization on large state spaces.

On the other hand, the DDPG algorithm also implement certain improvements that overcome the problems and challenges presented when using Deep Q-learning: instability of the learning process, generalization of the solution to multiple environments, and handling a mechanism to improve the exploration-exploitation trade-off. First, in order to solve the relatively unstable problem of learning the action-value function and optimize the gradient results significantly, target networks were modified to use soft target updates, rather than directly copying the weights. Then, the underlying weights are slowly updated by exponentially moving their average values using $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$ with $\tau \ll 1$.

A second improvement solved the difficulty of finding hyper-parameters that allow generalization across environments when the low dimensional feature vector observations have different physical units and the ranges. This issue is addressed by adapting a batch normalization technique proposed by [***], that minimizes the covariance shift during training by normalizing each dimension across samples in a mini-batch to have unit mean and variance. As a result, DDPG ensures that each layer receives whitened input in favor of learning effectively across many different tasks with differing types of units.

The third improvement tries to overcome the common challenge on how to manage exploration in reinforcement learning tasks with continuous actions spaces. The exploration-explotaition trade-off was managed independently from the learning algorithm by adding noise sampled from a noise process $\mathcal{N}$ to the actor policy μ and thus create an exploration policy $\mu'(s_t) = \mu(s_t|\theta_t^{\mu}) + \mathcal{N}$. Hence, any sampling process $\mathcal{N}$ can be plugged in to suit exploration on specific environments.

To evaluate the algorithm under physical environments with different levels of difficulty, they used the the Ornstein-Uhlenbeck[***] sampling process for adding exploration to the policy. The results demonstrated that even when harder tasks obtain poor Q estimates, DDPG is able to learn good policies across a variety of domains with continuous action spaces and using both low-dimensional feature vector and high-dimensional pixel inputs. Additionally, all the experiments were solved using fewer steps of experience than the DQN algorithm, showing that DDPG may be able to solve even more difficult problems. However, it requires of a large number of training episodes to find solutions and only works in environments with a small set of actions, so more robust model-free approaches should be proposed in order to tackle these limitation.

Later, **Deep Reinforcement Learning in Large Discrete Action Spaces** presented a new policy architecture, under the same actor-critic framework used in [***], that not only allows reinforcement learning methods to be applied to large-scale learning problems, and to operate efficiently with a large number of actions, but also can generalize over the action set in logarithmic time. The policy is then trained and optimized using DDPG. As a result, they obtained a more efficient algorithm that makes both learning and acting tractable in time and allows value-based policies to use the action features to reason about previously unseen actions.

The *Wolpertinger* architecture introduces two characteristics that previous approaches did not provide: 1)leverages prior information about the actions and embed them in a continuous space upon which the actor can generalize using a smooth function approximator, and 2)after the policy produces a continuous action within this space, it then uses an approximate nearest neighbor search to find the set of closest discrete actions in logarithmic time to finally select the highest valued action relative to a cost function. Overall, the algorithm is now able to handle large-scale reinforcement learning tasks by adding a sub-linear complexity relative to the action space and the ability to generalize over actions.

Moreover, the algorithm also avoids the heavy cost of evaluating all actions mainly by defining an efficient action-generating actor, and then using the critic to refine the actor?s choices for the full policy. The underlying function approximators are built using multi-layer neural networks and the resulting policy is trained with DDPG.

The action generation function $f_{\theta^{\pi}}(s) = \hat{\mathbf{a}}$ provides a proto-action in $\mathbb{R}^n$ for a given state s. $\hat{\mathbf{a}}$ is then mapped to the discrete action set $\mathcal{A}$ by applying the k-nearest-neighbor[3] function from equation 2.43, which returns the k closest actions in $\mathcal{A}$ by computing the L_2 distance between them. Even if g_k has the same complexity as the *argmax* in the Q action-value function, each step of evaluation of the L_2 distance is faster than a full value-function evaluation as the lookup is performed in logarithmic time.

$$g_k(\hat{\mathbf{a}}) = \arg \overset{k}{\underset{a \in \mathcal{A}}{\min}} |\mathbf{a} - \hat{\mathbf{a}}|_2 \tag{2.43}$$

Following the selection of the of k closest actions to $\hat{\mathbf{a}}$, the choice of the actual action to be applied to the environment is set according to the highest-scoring action obtained according Q_{θ^Q} defined in equation 2.44. Consequently, the full Wolpertinger policy π_θ with parameter θ, representing both the parameters of the action generation element θ_π and of the critic θ_Q, makes the algorithm significantly more robust to imperfections in the choice of action representation.

[3]The size of the generated action set k is task specific, and allows for an explicit trade-off between policy quality and speed

$$\pi_\theta(s) = \arg \max_{a \in g_k \circ f_{\theta^\pi}(s)} Q_{\theta^Q}(s,a) \tag{2.44}$$

The goal of this approach is to perform policy iteration by alternatively performing policy evaluation on the current policy with Q-learning, and then improving upon the current policy by following the DDPG configuration introduced in [***](previous paper) over f_{θ^π}, considering that the effects of g are a deterministic aspect of the environment. In this case, the critic is trained from samples stored in the replay buffer (which are generated by the full policy $\pi_{\theta^\pi} = g \circ f_{\theta^\pi}(s)$), whereas the policy gradient $\nabla_a Q_{\theta^Q}(s,a)$ is taken at the actual output â $= f_{\theta^\pi}(s)$. Finally, the target action in que Q-update is generated by the full policy.

The agent was evaluated on three environment classes: discretized continuous control, multi-step planning, and recommender systems. Specifically, the latter used a simulated recommendation system to demonstrate how the agent would perform on a real world large action space problem. Results showed that a subset of the full set of actions is sufficient in many tasks to converge and provide significant speedups. Therefore, the Wolpertinger algorithm can scale to real-world MDPs with large number of actions, but with an still existing issue of exploration when the agent needs to learn from scratch.

Chapter 3

Models

3.1 Naïve Model using Matrix Factorization

3.1.1 Reinforcement Learning model

The model is defined as a Model-free approach which its main objective is to find the optimal policy which maximizes the total future reward

State space: is the set of items recommended to the user

Action space: are the single items recommended at each timestep

Value function: rating feedback received after giving a recommendation.

At current iteration h+1, we have gathered h observed recommendation history Dh = (vi, ti, ri)h i=1. Collect user rating rh and update p(? | Dh)

3.1.2 Evaluation

*For insights and arguments, follow **Recommender systems survey**

Chapter 4

General Conclusions

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Appendix A

An Appendix About Stuff

(stuff)

Appendix B

Another Appendix About Things

(things)

Appendix C

Colophon

This is a description of the tools you used to make your thesis. It helps people make future documents, reminds you, and looks good.

(example) This document was set in the Times Roman typeface using $\LaTeX$ and Bib$\TeX$, composed with a text editor.

Bibliography

[1] Anne Author. Example Journal Paper Title. *Journal of Classic Examples*, 1(1):e1001745+, January 1970.