DEEP REINFORCEMENT LEARNING BASED INFORMATION RETRIEVAL

A Practice School Report submitted to
Manipal Academy of Higher Education
in partial fulfilment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

in

Computer Science & Engineering

Submitted by
Roshan Jacob Manoj
170905450
Under the guidance of

MR. ASHISH SRIVASTAVA
Scientist-D
CAIR
DRDO

&

DR. P. C. SIDDALINGASWAMY
Associate Professor - Senior Scale,
Computer Science & Engineering
Manipal Institute of Technology

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

July 2021

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Manipal
10/07/2021

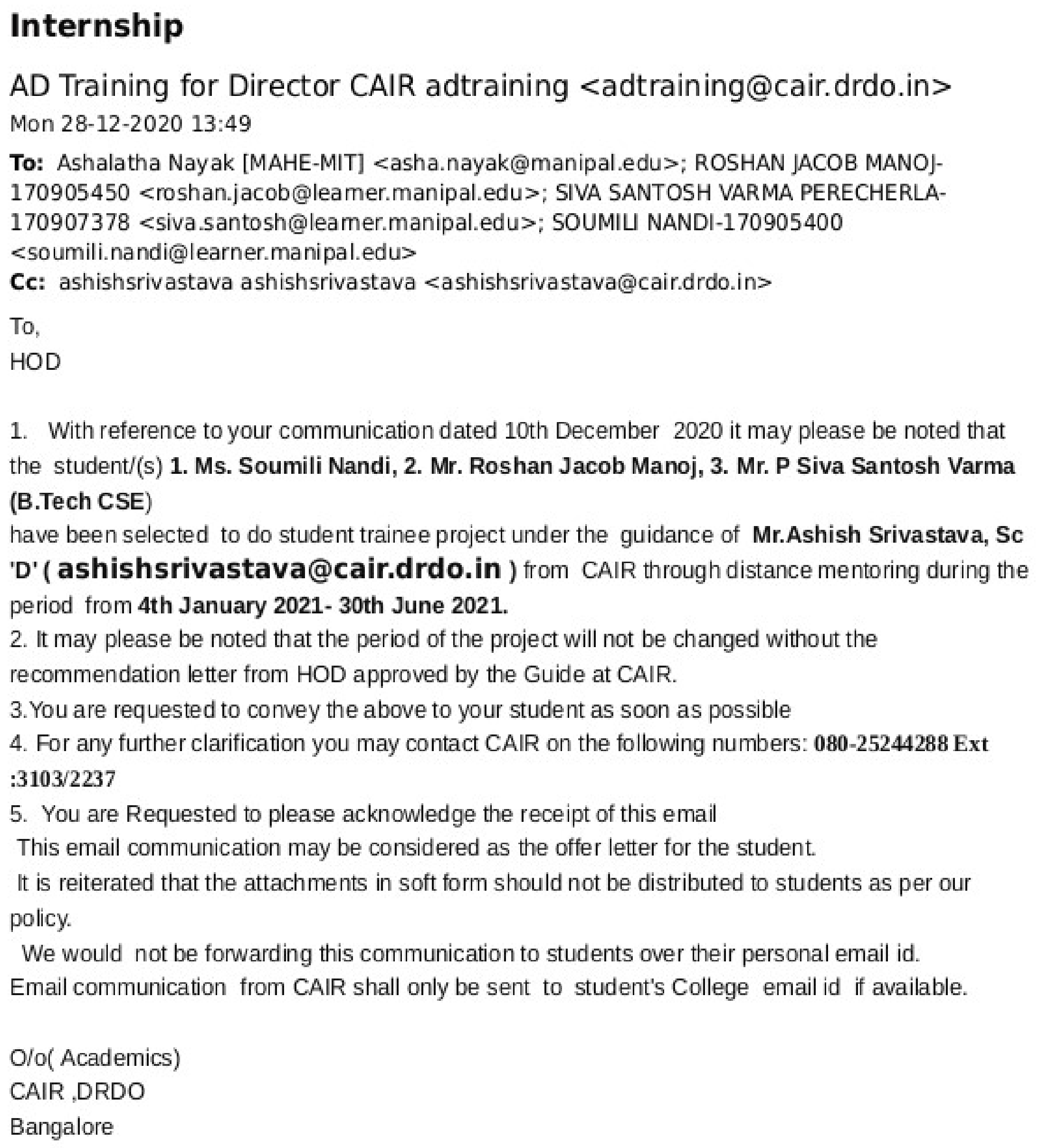
CERTIFICATE

This is to certify that the project titled **DEEP REINFORCEMENT LEARNING BASED INFORMATION RETRIEVAL** is a record of the bonafide work done by **ROSHAN JACOB MANOJ** (*Reg. No. 170905450*) submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology (B.Tech.) in **COMPUTER SCIENCE & ENGINEERING** of Manipal Institute of Technology, Manipal, Karnataka, (A Constituent Institute of Manipal Academy of Higher Education), during the academic year 2021.

Dr. P. C. Siddalingaswamy
Associate Professor - Senior Scale,
CSE Dept., M.I.T, MANIPAL

Prof. Dr. Ashalatha Nayak
HOD, CSE Dept.
M.I.T, MANIPAL

Offer Letter



Internship

AD Training for Director CAIR adtraining <adtraining@cair.drdo.in>
Mon 28-12-2020 13:49

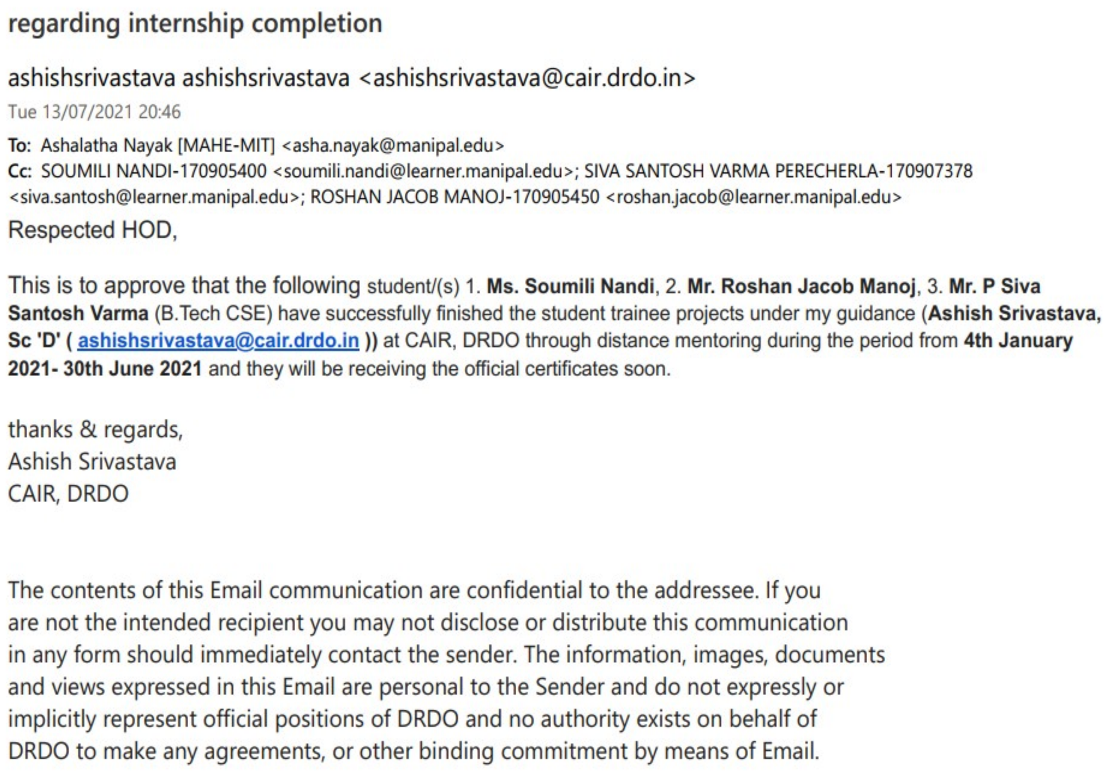
To: Ashalatha Nayak [MAHE-MIT] <asha.nayak@manipal.edu>; ROSHAN JACOB MANOJ-170905450 <roshan.jacob@learner.manipal.edu>; SIVA SANTOSH VARMA PERECHERLA-170907378 <siva.santosh@learner.manipal.edu>; SOUMILI NANDI-170905400 <soumili.nandi@learner.manipal.edu>
Cc: ashishsrivastava ashishsrivastava <ashishsrivastava@cair.drdo.in>

To,
HOD

1. With reference to your communication dated 10th December 2020 it may please be noted that the student/(s) **1. Ms. Soumili Nandi, 2. Mr. Roshan Jacob Manoj, 3. Mr. P Siva Santosh Varma (B.Tech CSE**)
have been selected to do student trainee project under the guidance of **Mr.Ashish Srivastava, Sc 'D'** (**ashishsrivastava@cair.drdo.in**) from CAIR through distance mentoring during the period from **4th January 2021- 30th June 2021.**
2. It may please be noted that the period of the project will not be changed without the recommendation letter from HOD approved by the Guide at CAIR.
3.You are requested to convey the above to your student as soon as possible
4. For any further clarification you may contact CAIR on the following numbers: **080-25244288 Ext :3103/2237**
5. You are Requested to please acknowledge the receipt of this email
This email communication may be considered as the offer letter for the student.
It is reiterated that the attachments in soft form should not be distributed to students as per our policy.
We would not be forwarding this communication to students over their personal email id.
Email communication from CAIR shall only be sent to student's College email id if available.

O/o(Academics)
CAIR ,DRDO
Bangalore

Project Completion Letter



regarding internship completion

ashishsrivastava ashishsrivastava <ashishsrivastava@cair.drdo.in>

Tue 13/07/2021 20:46

To: Ashalatha Nayak [MAHE-MIT] <asha.nayak@manipal.edu>
Cc: SOUMILI NANDI-170905400 <soumili.nandi@learner.manipal.edu>; SIVA SANTOSH VARMA PERECHERLA-170907378 <siva.santosh@learner.manipal.edu>; ROSHAN JACOB MANOJ-170905450 <roshan.jacob@learner.manipal.edu>

Respected HOD,

This is to approve that the following student/(s) 1. **Ms. Soumili Nandi**, 2. **Mr. Roshan Jacob Manoj**, 3. **Mr. P Siva Santosh Varma** (B.Tech CSE) have successfully finished the student trainee projects under my guidance (**Ashish Srivastava, Sc 'D' (ashishsrivastava@cair.drdo.in))** at CAIR, DRDO through distance mentoring during the period from **4th January 2021- 30th June 2021** and they will be receiving the official certificates soon.

thanks & regards,
Ashish Srivastava
CAIR, DRDO

ACKNOWLEDGEMENTS

With immense pleasure and gratitude, I would like to extend my sincere and thanks to all the individuals who helped me during my practice school training and made it a success. I perceive this opportunity as a big milestone in my career development. I will strive to use gained skills and knowledge in the best possible way to reach my career goals.

I am using this opportunity to express my deepest gratitude and special thanks to Mr. Ashish Srivastava, my supervisor, without whom this project would not have been possible, who also took the time out to guide me with the best resources and continually mentor me. Despite these unprecedented circumstances due to which the internship was held remotely, he ensured, with his timely replies, that I face no roadblocks. Sincere thanks to all the authorities at the Centre for Artificial Intelligence & Robotics, DRDO, Bangalore, who allowed me to implement the project.

I am thankful to my internal guide Dr. P. C. Siddalingaswamy for his revisions to the midterm and thesis report, and for scheduling the timely evaluations. I would also like to thank the Director of MIT and the Head of Department (HOD) of Computer Science Engineering, Prof. Dr. Ashalatha Nayak, for her unending support throughout my time till now at the institute. I would also like to thank very knowledgeable faculty members in the Computer Science and Engineering department who helped make my concepts clear from day one, which led to the successful completion of my training project.

ABSTRACT

The research work done focuses on the use of Reinforcement Learning to perform ranking on a dynamic search system. This involved a 2 part process, one that involved the design and architecture of the search system and another that considered the RL ranking problem separately and improvements on the same. Most of the information retrieval system applications such as recommendation engine, search engines, online advertisements make use of the user preference in order to provide a more accurate and reliable result. These systems are often made using machine learning algorithms that are either supervised or unsupervised. In recent developments, reinforcement learning which is another machine learning algorithm, has gained acknowledgement on various environments showcased by OpenAI. We start the report with a design of a dynamic search model that uses Reinforcement Learning algorithm Such a model incrementally updates the size of the candidate set of documents over time and takes the user feedback with respect to the query searched. We reference the RLIRank[10] paper for this. Due to lack of a simulator that provides documents continuously to the user and records feedback, we designed our own GUI simulator. Post this, we test the Reinforcement Learning model that performs ranking as a standalone on the MQ2007 and MQ2008 datasets [32]. This is done to verify the base implementations of the MDPRank [13] and PPGRank [17] paper based on which we performed further experiments. The experiments involved the extension of the MDP formulated with value based algorithms and actor-critic based algorithms. Value based algorithm like Deep Q Network and actor-critic based algorithms like Q-actor critic, variations of Advantage Actor critic, Proximal Policy Optimization algorithms were implemented next and discussed in the report. The results were then tested and analysed across with respect the LETOR 4.0 [32] datasets and their respective baselines. The dynamic model uses the Microsoft's Reading Comprehension [30] dataset and presents a demo to the user along with the implications of the Reinforcement Learning model. To conclude, we have built a dynamic search model with concepts taken from information retrieval systems and reinforcement learning algorithms and previous papers, which can be further extended by other researchers to perform a more real dataset scenario of the application of reinforcement learning and also to evaluate their own algorithms using the model. The implementation has majorly been done in Python with the help of Pytorch library and we make extensive use of libraries like Numpy and Gensim for the implementation.

LIST OF TABLES

LIST OF FIGURES

Table of Contents

CHAPTER 1

INTRODUCTION

1.1 Introduction

This chapter will be providing a general overview of the area of work focused on, which is using Reinforcement Learning in an Information Retrieval system (section 1.2) along with the current state of the field and its limitations in section 1.3. Later on, it also provides the motivation for the research done and what is hoped to have been achieved and have been contributed in section 1.4. Further it also discusses the objectives of this project work in section 1.5. Thereon, briefly mentioning the target specifications used in section 1.6. along with the schedule followed to accomplish the objectives of the project in section 1.7. Concluding this chapter with section 1.8, it further briefly mentions how the remainder of the project is structured.

1.2 Introduction to the area of work

Search engines, E-commerce applications, online advertisements, recommendation systems have become a necessity when surfing the internet. From looking for answers of general questions to using the internet for buying products, the average user relies on the results provided by the system in order to access the internet. In turn, the results have to be fast, accurate and reliable. Such is the area of information retrieval systems. While there exists a never-ending amount of information available on the internet, the retrieval system allows for the user to access that information in a simple and efficient manner. Evidently, ranking plays an important role in such a system. While ranking is performed on a list of documents taking into consideration various factors which includes but is not limited to the user preference, it is still far from perfect. Such a module that exists in the system is called Learning to Rank (LTR). An information retrieval system must not aim to be relevant to all users, rather, it must learn to personalize its relevance in accordance to the user behaviour. To promote research in this direction various conferences like the Text Retrieval Conference (TREC)[37] Neural Information Processing Systems (NIPS)[27], Special Interest Group on Information Retrieval (SIGIR)[28] and International Conference on machine learning(ICML)[29] have held workshops.

With the recent successes provided by OpenAI papers that focus on Reinforcement Learning which is used in various environments, that learn without the knowledge of the environment provided to it, it is only inevitable that such a technique would be applied to real case scenarios such as that in information retrieval systems. Being another type of machine learning algorithm, reinforcement learning aims to solve environments with the help of experience (trial and error). Hence it learns from the interaction with the environment rather than on a supervised or unsupervised set of labels.

Our area of work deals with applying the reinforcement learning method to the learning to rank paradigm of information retrieval system. Further aiming to achieve a dynamic search model that relies on the user's interaction with the agent hence providing a natural extension of the idea of reinforcement learning problems. Hence the system's design and development takes not only the query and the documents relation, but also the users preference which makes it personalized to the requirements of each user.

1.3 Brief present day scenario with regard to the work area

Past research has successfully applied machine learning to the learning to rank task, however most of the past research in this domain has been limited to supervised datasets or rather datasets with labels. For instance, the Bing search engine is said to be using the RankNet [3] algorithm which harnesses the power of neural networks in order to be able to learn to rank. The Apache open source Solr search engine makes use of machine learning based ranking as of 2017. Further applications of this are seen in ElasticSearch, which is used in various applications like Netflix, Slack, etc.

Recent research papers aim to make use of Reinforcement Learning to perform information retrieval and Learning to Rank based on user feedback [10,11]. Some research has focused on applying RL to specifically LTR settings and evaluating them on supervised datasets to provide a fair comparison to previous machine learning methods. RL based methods have been proven to provide state of the art results for some datasets [13,17]. This is further explained in chapter 2.

1.4 Motivation to do the work

It can be observed that the application of Reinforcement Learning in Information Retrieval hasn't been exploited in terms of both effectiveness and transparency with simplicity for extension. Further, it can be noticed that there are two main research branches in this domain, one branch that specifically deals with the application of RL paradigms to supervised learning datasets and proving its effectiveness but does not include the users feedback and hence is not personalized in nature. Another branch that focuses on trying to apply a dynamic learning paradigm to the Learning To Rank task using Reinforcement Learning, however this branch is still in its infancy and most work so far is specialized without any scope of verification or reproduction and also lack of proper justification.

Having been inspired by the work so far, that is, in the application of Reinforcement Learning in IR systems, it led to the likeliness of contributing to the progress in this field by building an effective, robust and transparent IR system that improves the RL algorithm used for LTR using the supervised learning datasets and then add the concepts of dynamic ranking to account for user preferences to form a complete RL based IR system.

1.5 Objective of the work

The objectives of the project were:

- Learning Reinforcement Learning algorithms theoretically
- Researching on papers that provide ways in which “Learning to Rank” is done by Reinforcement and Machine Learning Algorithms.
- Researching on papers that construct a dynamic search model and preferably use reinforcement learning techniques for the same.
- Implementing various reinforcement learning algorithms for ranking from papers researched.
- Using the implementation to generate insights
- Improving on the results by trying different experiments and heuristics.
- Construction of an end to end model that performs dynamic search with the use of reinforcement learning model.
- Verifying and comparing the results obtained with other state of the art methods and implementations.

1.6 Target Specifications

- Input: Take the entire corpus of documents from the dataset as input. Simulate user(s) who can give feedback on the displayed ranking.
- Intermediate: Retrieve a subset of the corpus and convert documents into embedding representation by projecting them to latent space and then give them as input to the RL ranking agent.
- Output: Top k relevant documents are retrieved according to previously accounted user preferences and quality of document. RL agent learns from the user feedback.
- Target user: This research work can directly be used for applications that have a need for an Information Retrieval System that has to be dynamic with respect to user preferences.
- Duration: The proposed project is expected to finish in 6 months.

1.7 Project schedule

The schedule in the internship can be summarized as follows:

January 2021 - February 2021

The internship began with the study of the rudimentary topics required to understand the project. The domain of our internship was in machine learning and hence the reason to start with the infamous “Andrew NG course on Machine Learning”, which was further extended with the theory essential to understand mathematics and derivations, essentially the subject of probability and statistics. To this, Schaum’s Outline of Probability & Statistics [26] was referred to.

Reinforcement Learning basics were made concrete using the lecture series by David Silver. Along with this, various research papers about the original Reinforcement Learning algorithms were also read. Simultaneously, there were one-on-one sessions with our mentor to understand the Reinforcement learning concepts to the core. The topics covered were: RL definition, MDPs, Markov chains, Bellman Equations, Value based methods, Policy gradient, Actor critic methods among many others.

March 2021 - April 2021

Implemented some of the algorithms learnt theoretically using Pytorch on environments provided by Gym. This involved performing experiments on the Lunar Lander environment and Cartpole environment. This helped us understand the algorithms even better and form better intuitions.

During this period, extensive research on papers that constructed a dynamic search model using reinforcement learning were also conducted. Further, a literature survey on the “Learning to Rank” based algorithms that used Reinforcement Learning were performed. During these months a simulator with a GUI interface was built incrementally. On choosing a research paper that suited the need of performing ranking, implementation of the paper was done. This allowed us to verify the results and aim for future improvements on the same.

May 2021 - June 2021

These months were used in exploring and experimenting various ideas to improve the results obtained by previous implementations.Experiments on Learning To Rank benchmark datasets using Reinforcement Learning were also performed. Implemented, fine-tuned and analysed multiple modern Reinforcement Learning algorithms and their application to the standard LETOR datasets provided by Microsoft. Further, the design of the Information Retrieval system that performed dynamic search using reinforcement learning model for ranking based on the user preference was also architected and finalised.

1.8 Organization of the project report (chapter wise)

The remainder of the thesis has been organized into the following sections: Background Theory (refer 2), Methodology(refer 3), Result Analysis(refer 4), and Conclusions and Future Scope(refer 5). In Chapter 2, a detailed literature review and a study of the background knowledge necessary to conduct the experiments in this project are performed. It highlights the research gaps and the premise on which this project was conceived. Following this, a description of the methodology followed in the project is provided in Chapter 3, including any tools used, experiment setup, or theoretical knowledge inferred. The result section, in chapter 4, is a brief description of the end-product and the experimental evaluation of this project. In the final section, chapter 5, the report is concluded by explaining our observations and learnings from the project and then it proceeds to outline the future scope of the project. This section will be the focus of this work for future researchers/engineers.

CHAPTER 2

BACKGROUND THEORY / LITERATURE REVIEW

2.1 Introduction

This section aims to understand the paramount intuition and the theoretical knowledge required to understand the project. This pertains to the inclusion of an overview of the information retrieval system that shall be targeted first. Further it focuses on the crux of the project which is reinforcement learning, the intuition behind it and the various algorithms involved. Complimentary to this, it will also discuss in brief the various approaches taken by previous works in the same domain and the ideas that were extended further in the project. This section aims to deliver the knowledge, the background theory and the previous work done, which is necessary to formulate the solution that is explained in Chapter 3.

2.1.1 Suggested Reading

With the help of the university recommended resources, good practices were followed during the group's interactions, meetings and work. This project involved members with different backgrounds who collaborated into the work equally and in an impartial manner. With the help of [23], we followed the practice of involving reusable structures into the project everyday. With the help of [24] and [25] the practices of ethics and inclusion were used in the research domain. It allows for an amalgamation of researchers belonging to various cities, genders and profiles to work towards a common goal of providing a reliable and an extensive approach for further work involved in this field.

2.2 Introduction to the project

The project title, "Deep RL based Information Retrieval", defines the problem of reinforcement learning techniques to construct an information retrieval system wherein, an end to end dynamic information retrieval system is showcased, which essentially, not only focuses on ranking documents accurately on the basis of the query provided but also on the basis of the user's preference. The system aims to achieve a better ranked list for the user, based on the interactions provided by the user to the system. Hence, a natural appending work would be to test each of these Reinforcement Learning techniques used with the available baselines on a similar dataset. This leads to experimentation with a plethora of Reinforcement Learning algorithms and design choices that are required to be able to construct such a system.

2.3 Literature review

An information retrieval system primarily deals with searching for a relevant piece of information amidst an abundant amount of data present in the database of the system. Hence the task of an information retrieval system is two-fold. One, to retrieve a set of documents from the database and second, to use the retrieved set of documents to rank which is coined as Learning to Rank (LTR).

Previously in this field, many machine learning models have been constructed to perform ranking of the documents and they can be briefly classified into three approaches. First, the pointwise approach, second the pairwise approach and lastly, the listwise approach. There have also been models constructed that constitute a hybrid between these approaches. In the pointwise approach each document is evaluated separately without any comparison or ordering done with respect to another document. Hence, it makes use of a single document, which is further provided to the machine learning model that learns on the basis of the relevance label defined for the document with respect to the query [1, 2, 34]. In the pairwise approach[3,4,5], two documents are compared in order to take into consideration the relative ordering with respect to the ground truth. The learning is decided on the basis of the pair that minimizes the wrong relative order. Lastly, the listwise approach [6,7,8] makes use of probabilistic models in order to reduce the ordering error achieved when ranking the entire list of documents. To evaluate the various approaches, information retrieval evaluation measures [22] are used in order to assess how well the documents are ranked. One of the measures used is the Normalized Discounted Cumulative Gain (NDCG). This measure sprouts from the discounted cumulative gain (DCG).

$$DCG_k = \sum_{i=1}^{k} \frac{2^{rel_i}}{log_2(i+1)} \quad (2.1)$$

$$NDCG_k = \frac{DCG_k}{IDCG_k} \quad (2.2)$$

In equation 2.1 the k stands for the top k documents used for the measure, rel_i is the relevance label of the document at i^{th} position. In equation 2.2 the $IDCG_k$ is the best DCG value possible for the top k documents. The ratio of these 2 terms provide the normalized DCG value.

The primary paper used to understand and have an idea of the dynamic learning to rank framework is the “RLIRank” paper [10]. This paper made it possible to construct a simulator that allowed for dynamic search. The paper demonstrates how a dynamic search model would be built in order to adapt to a reinforcement learning model. The model is as shown in Fig. 2.1.

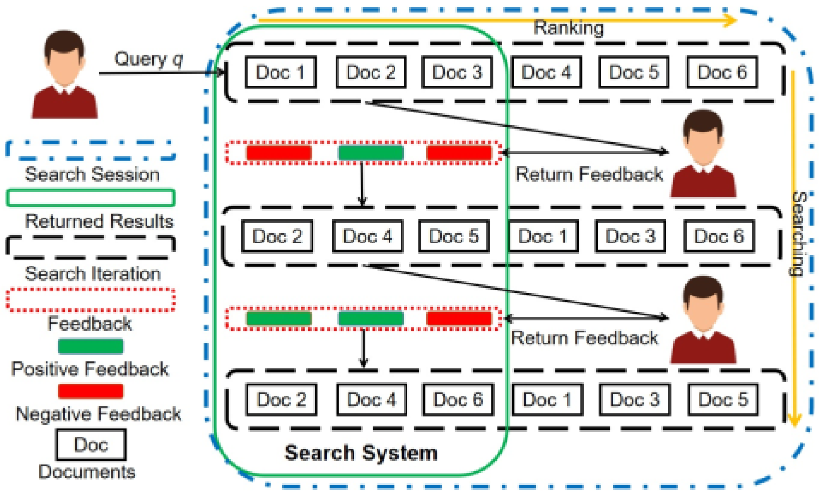


Fig 2.1 User Feedback based Dynamic Search Model [10]

The paper makes use of a search model that provides a list of documents to the user and incorporates feedback from the user into the Information Retrieval system. Essentially, a list of candidate documents are provided to the agent. The agent acts as the search system and performs the action of ranking the documents. These ranked results or search results are activated when the user fires a query. Once the results have been shown to the user a feedback is taken from the user. Based on the feedback the query is reformulated and a new query is made. The process is then repeated. The Reinforcement Learning agent makes use of a deep value network that applies a stepwise approach to train the model. The value network is implemented using a stacked recurrent neural network model, which is tuned on the basis of these search iterations.

Another paper that explains the concept of dynamic search and in fact uses it in a real world scenario is the "Reinforcement Learning to Rank in E-commerce Search Engine" paper[11]. This paper focuses on using search sessions in order to adapt the ranking paradigm to an E-commerce search engine. With the help of TaoBao application, the paper demonstrates a multi-step ranking setting. They provide a modification to the deterministic policy gradient algorithm [12] and call it "deterministic policy gradient algorithm with full backup estimation"[11]. The users are simulated by a probabilistic model and their effectiveness is tested on the real application to showcase the effectiveness of the algorithm.

Further on, the "Reinforcement Learning to Rank with Markov Decision Process" [14] was referenced. In this paper, the author presents a novel algorithm to perform ranking and proved to outperform previous state-of-the-art results. They provide a formulation of a Markov Decision Process that works for the ranking paradigm. They represent the MDP as a tuple (S, A, T, R, π) where the states S consists of a list of state defined for each time step t paired with the documents that are left to be ranked denoted by X_t. The actions A is a set of actions that the agent performs on the state s_t in order to select a document to place at position t+1. The transition T provides a mapping from the current state to the next state using the action chosen at the current state to remove the document chosen from the set of remaining documents. The agent is incentivized with rewards calculated from the DCG evaluation metric.

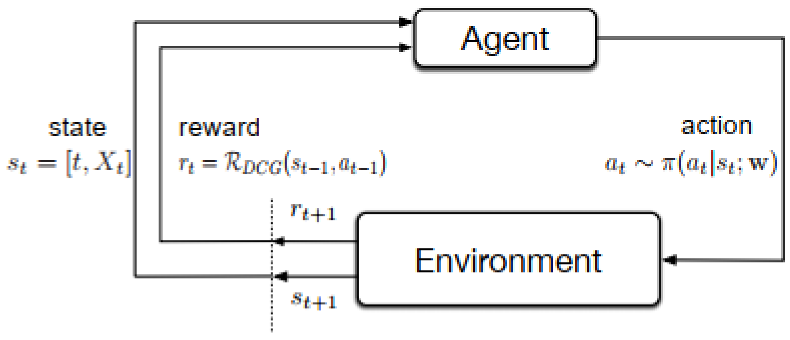

Agent
state
$s_t = [t, X_t]$
reward
$r_t = \mathcal{R}_{DCG}(s_{t-1}, a_{t-1})$
action
$a_t \sim \pi(a_t|s_t; \mathbf{w})$
r_{t+1}
Environment
s_{t+1}

Fig 2.2 Interaction of the agent with the environment in MDPRank [13]

Using the REINFORCE[9] algorithm the parameters of the agent are learnt with the help of the training dataset. This model is further implemented and used to reproduce the results and act as the base for our approach to understanding reinforcement learning and also improvements. The model however, does not take the relative ordering of the documents into consideration, suffers from high variance and is not a neural network implementation which might cause it to be ineffective when learning a complex ranking function.

To perform experiments on the chosen Markov Decision Process for our setting, a literature survey on the various reinforcement learning models were done. In a value based approach, the aim is to be able to approximate the Q-value using a neural network. This is called Q-learning[9, 35]. However, such a learning suffers overfitting in cases where the inputs or the outputs are correlated and not independent. Another issue with Q-learning is the non-stationarity of the targets in the loss-function. This is an issue faced in general with any implementation of reinforcement learning with neural networks.

To solve these issues, an algorithm called the Deep Q Network (DQN) [9,14] was introduced. It made use of an experience replay memory which acts like a buffer in the sense that it stores the transitions into small batches of the data referred to as “minibatch transitions”. Using stochastic gradient descent on the experience replay memory, it randomly samples minibatches and hence trains the network. Once the buffer has reached its full capacity, the new transitions shall replace the old transitions. To tackle the issue of non-stationarity, the targets are computed with respect to the old parameters of the neural network and then updated in small time intervals with the new learned parameters. This provides better performance and better estimate of the value function. This method is a value based method and is quite prominent in the Deep Reinforcement Learning literature.

Unlike value based methods where the policy is generated from the value function, policy based methods aim to directly learn the parameters of the policy. The advantage of such an approach is that compared to value based methods it has better convergence, however the convergence leads more often to a local minima than a global minima. These methods also suffer from high variance. One of the oldest policy based method is the REINFORCE [9] algorithm. In this algorithm, based on the policy and its parameters an entire episode is sampled. Based on the sampled episode, the return and the score function is calculated for each time step. This is then used to update the policy parameters. Finally, the process is repeated until convergence is achieved.

Actor Critic methods make best use of both, value based methods and policy based methods. The actor learns to approximate the policy using the critics estimate while the critic learns to estimate the actor's policy by minimizing the error with the true Q-values [35, 15]. These algorithms provide faster convergence and low variance with better convergence properties. They may share or use separate parameters. A direct correlation of this method is the advantage actor critic algorithm. As the policy gradient theorem states in equation 2.3,

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta log\, \pi_\theta(s,a)\, Q^{\pi_\theta}(s,a)] \tag{2.3}$$

where J is the policy objective function parameterised by θ. The advantage actor critic methods approximate the advantage of an action while introducing the value of the state as the baseline. Hence the equation 2.3 becomes equation 2.5,

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s) \tag{2.4}$$

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta log\, \pi_\theta(s,a)\, A_w(s,a)] \tag{2.5}$$

where, $A_w(s,a)$ is called the advantage estimate parameterised by the critic's parameters w. Advantage actor critic(A2C) is merely a synchronous version of asynchronous advantage actor critic(A3C)[16] and was made prominent in the field due to its simplicity and solving ability.

Asynchronous advantage actor critic introduces parallelism by using a master network that communicates with other workers. Each of these workers have a copy of the parameters of the master and interact separately with the environment. The workers then trigger the updates when it deems necessary. This paradigm of parallelism decorrelates the agents data into a more stationary process and allows us to deploy deep neural network approximators to a large class of on-policy RL algorithms. However, since the initial copy of the parameters are different along with enabling exploration, the workers use policies of different versions. Because of this, the aggregate update is not optimal. To tackle this problem A2C synchronizes by waiting for the workers to complete before updating the masters parameters. Hence ensuring that in every iteration the workers all start with the same policy. This provides a more generalized update and evidently faster convergence than A3C. With regards to the performance, A2C has shown to give the same or better performance than A3C with the correct utilization of GPUs. Another interesting addition to A3C is the concept of entropy regularization which is added as an additional term in the objective function itself. It enforces exploration in on-policy methods which proves to be useful for convergence in some situations.

2.3.1 Recent developments in the work area

The RLIRank paper [10], as reviewed before, is a paper published in 2020. Considering its approach our major motivation for the information retrieval system was taken from this paper. However, the reinforcement learning agent's Markov Decision Process formulation was enabled by the MDPRank paper[13].

A more recent extension of the MDPRank paper is the Reinforcement Learning to Rank with Pairwise policy gradient paper (PPGRank) [17]. This paper aims to tackle the issues faced in the MDPRank paper. While the MDPRank paper performed ranking in a listwise approach, PPGRank paper aimed to perform the ranking in pairwise approach. This is to ensure that relative ordering of the documents were taken into account. The paper further follows along to prove that PPGRank provides a low variance and low bias estimation of the gradients. PPGRank aims to estimate gradients on the basis of intra-query comparisons between 2(or pairwise) samples of the document list.

Similar to MDPRank, but instead of sampling an episode for a query once, PPGRank samples 2 episodes for each state starting from the timestep of the current state to the end of the trajectory. This increases the computational speed time with respect to the MDPRank paper, but allows for a lower variance with faster and better convergence. The paper further proves that the extension of the MDPRank, i.e. PPGRank can be considered as a variation of the REINFORCE with baseline policy gradient algorithm. Even so, the paper fails to make use of neural networks in its implementation due to which it cannot learn a complex ranking function.

While there are constant advancements happening in the Reinforcement Learning domain, one potential algorithm stands out. Since the time of its release it has been used as the baseline for various environments due to its success. The algorithm is the Proximal Policy Optimization(PPO) Algorithm released in 2017 [18]. The PPO algorithm aims to allow for a trust region update but without the complicated methodology and by simply using the clipped surrogate objective. It makes use of the surrogate objective from TRPO and also introduces a clipping. The clipping restricts the importance sampling weight from becoming too large or too small. This allows for a more robust control of the policy and provides a similar performance than that achieved by Trust Region Policy Optimization (TRPO) [19]. The algorithm as mentioned in the paper is given below.

Algorithm 2.1 $PPO, Actor - Critic\ Style$ [21]

```
1  for iteration = 1,2 ... do
2      for actor = 1,2, ... N do
3          Run policy $\pi_{\theta_{old}}$ in environment for T timesteps
4          Compute advantage Estimates $\widehat{A_1}, \dots, \widehat{A_T}$
5      end for
6      Optimize surrogate L wrt $\theta$, with K epochs and minibatch size $M \leq NT$
7  end for
```

The official algorithm makes use of parallel actors starting from the same initial state. Firstly, an actor and a critic is initialized with random weights. Starting with n workers in parallel, each worker collects T step transitions with the policy using the old parameters, and computes the advantage estimates. Using the minibatch concept, M of these transitions are each considered as a minibatch. The actor is then trained to maximise the clipped surrogate objective, while the critic minimizes the error for each of these mini batches that are run for K epochs and finally updates are performed. The PPO algorithm has successfully improved the scores of some ATARI environments, however it does not provide any convergence guarantee.

2.3.2 *Brief background theory*

To understand the work and to explain the intuition used better, this section will be discussing the background theory in brief.

This section will first discuss a semantic model that was used in order to overcome certain limitations of the system. The semantic model provides a method that could generate paragraph or document embeddings for documents using either Distributed Memory (DM) or Distributed Bag of Words (DBOW) technique. This model ensures that similar words with similar meanings are represented closely in the embeddings space and also that there are multiple degrees of similarity. Since this model (Doc2Vec [21]) is an extension of another model (Word2Vec[20]), it is better to briefly look into Word2Vec to gain a better understanding of Doc2Vec.

The Word2Vec algorithm provides a numerical representation of each word with the help of feature vectors. This is essentially created with either the continuous bag of words (CBOW) model or the Skip-Gram model. In the former, a sliding window is used in order to predict the missing word from the words surrounding it. All of these are done numerically and hence training in such a way provides a numerical representation of the words. The latter is the more accurate approach when the words in the document are not as frequent. With the Skip-Gram model, the word is used to predict the words surrounding it. This is a slower, but much more effective technique than CBOW (depending on the dataset).

Doc2Vec expands on the idea by considering documents instead of words. This can be done with the Distributed Memory version of paragraph vector (PV-DM) or with the Distributed Bag of Words version of Paragraph Vector (PV-DBOW). In PV-DM, extending from the CBOW model, the authors introduced a feature vector that was unique to the document (Document ID). Then word2vec was performed on the words, however, the training happens simultaneously on the added document unique feature vector. Hence training the document vector which holds the numerical representation of the document. The PV-DBOW version works similar to Skip-Gram model, but instead of predicting the words based on a word that is taken as input, the words are predicted based on a document ID that is taken as input. PV-DM is known to work better, however the author suggests using a combination of the 2 techniques.

Further, this section will discuss some basic terminologies and understanding of reinforcement learning which is core to the system discussed in Chapter 3.

Reinforcement learning is a type of machine learning algorithm that learns from experience rather than learning to fit based on a training set. It learns through its interaction with the environment by selecting actions that are influenced by its past experience which is called exploitation and by the new choices taken which is called exploration. To evaluate and to learn, the Reinforcement learning agent makes use of rewards which tells the agent about the outcome of the action taken. The agent's goal is hence to learn to select those actions that maximise not only the immediate rewards, but also the rewards that are achieved in the future.

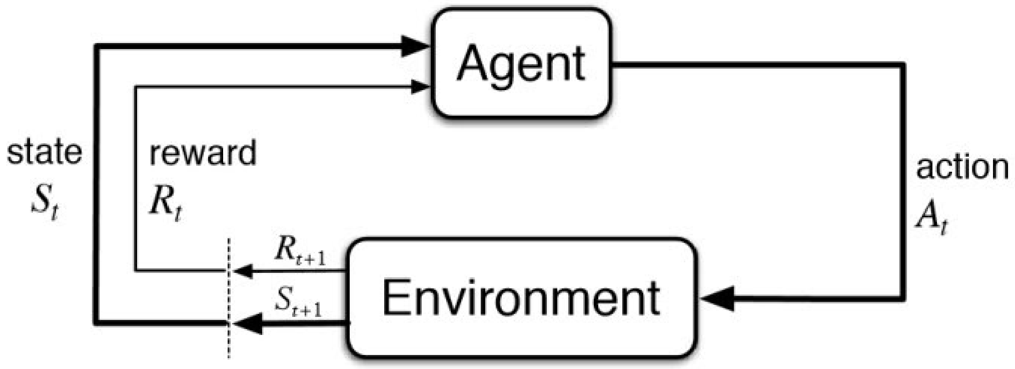


Figure 2.3. Agent Environment interaction in Reinforcement Learning. [9]

As shown in the above figure, in reinforcement learning there are two entities between which the interaction takes place. The agent and the environment. At each discrete time step the agent receives the next state (or initial state for the first time step) and the reward generated by taking an action (no reward for the initial state). The agent performs an action that is used by the environment to update to the next state and generate a reward for. This process is then repeated. The agent can otherwise be called the learner while the environment can be the problem that is to be solved. Intuitively, anything that the agent cannot change, becomes a part of the environment. The process by which the agent takes a set of actions which causes transition of the state, collected into a set of states, and generates rewards together makes up a representation of the process which is given by the Markov Decision Process (MDP).

Essentially, the MDP is a tuple $< S, A, R, P, \gamma >$ where, S denotes a set of states, A is a set of actions the agent can take, R is the reward collected by the agent on performing those actions (equation 2.6), P is the transition probability matrix (equation 2.7), and γ is the discount factor in the range of [0,1]. It also defines a policy π which indicates the agent's behavior. The policy is given by the equation 2.8.

$$R_s{}^a = E[R_{t+1} | S_t = s, A_t = a] \tag{2.6}$$

$$P_{ss'}{}^a = P[S_{t+1} = S' \mid S_t = S,\ A_t = a] \tag{2.7}$$

$$\pi(a|s) = P[A_t = a \mid S_t = s] \tag{2.8}$$

The goal of the agent is to hence maximise the expected returns given by equation 2.9,

$$G_t = \sum_{k=0}^{\text{inf}} \gamma^k * R_{t+k+1} \tag{2.9}$$

This allows the definition of two functions, the state-value function $V_\pi(s)$ and the action-value function $Q_\pi(s, a)$. The state-value function tells the agent how good a particular state is, whereas the state-action value function tells how good an action taken in a particular state under a policy π is. The state-value function is numerically the expected return taken from state s then following a policy π. The action-value function is the expected return of taking action a in state s and then following the policy π. Their respective equations are as given in equation 2.10 and equation 2.11.

$$V_\pi(s) = E_\pi[G_t \mid S_t = s] \tag{2.10}$$

$$Q_\pi(s, a) = E_\pi[G_t \mid S_t = s, A_t = a] \tag{2.11}$$

There are multiple ways in which an Reinforcement Learning agent can be classified. Based on its components, the RL agent can primarily be classified into 2 types - Model-Based and Model-Free as shown in fig 2.4

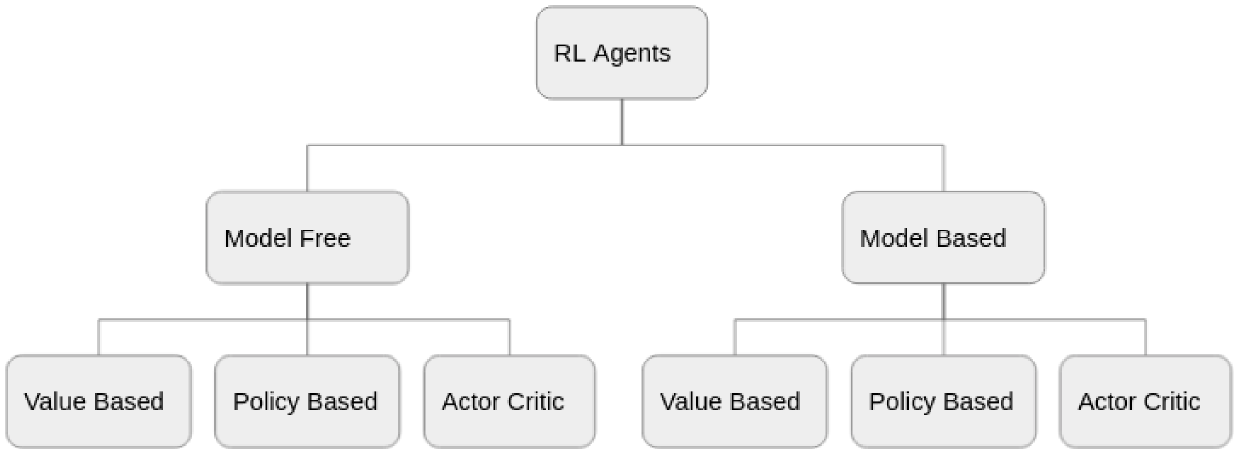


Fig 2.4. Types of RL Agents

In Model Free RL, the agent need not know the inner workings of the model or the system. However, in Model Based RL, the cost function needs to be defined post which, the optimal action is taken using the model directly. In both these classifications the agents are divided into mainly 3 categories which are, value based method, policy based method and actor critic. In brief, value based methods computes just the value function (could be state-value function or the state-action value function), and is hence an example of an off-policy algorithm. While value based methods aim to find the estimate of the optimal value function from which the optimal policy is achieved, policy based methods directly target achieving the optimal policy. Actor critic methods aim to achieve the best of both worlds, by learning both - a value function and a policy. Reinforcement learning can also be categorized into a planning problem or a reinforcement learning problem based on the environment. If the model of the environment is unknown, then it is a reinforcement learning problem. In such case, the agent interacts with the environment and improves its policy. If the model of the environment is known, then it is a planning problem as the agent performs computation with the model in order to improve the policy.

Most of the algorithms used to solve an RL problem as described earlier in literature review, try to find the optimal policy $\pi_*(a|s)$. Having the optimal policy ensures that rewards obtained are maximised. As mentioned earlier, value based methods estimate the optimal value function which in turn implicitly provides the optimal policy (equation 2.12). Policy based methods directly optimize the policy $\pi\ (a|s)$ to give the optimal policy $\pi_*(a|s)$.

2.4 Summarized outcome of the literature review

In summary, the literature review provided insights on how an information retrieval system is to be architected. The evaluation measures used, the various approaches taken to perform learning to rank and how the reinforcement learning paradigm fits into the setting can be understood from the literature review. It is paramount to be able to differentiate each of the methods used in reinforcement learning in order to find a method that best suits our setting. It could be concluded from the previous papers used to perform ranking with reinforcement learning, how the MDP is to be formulated and the literature survey of the various algorithms provide an idea of what kind of modification and adaptation could provide better scores. Further the semantic model used overcame certain limitations that were faced while implementing the setting, which is further discussed in Chapter 3.

2.5. General analysis

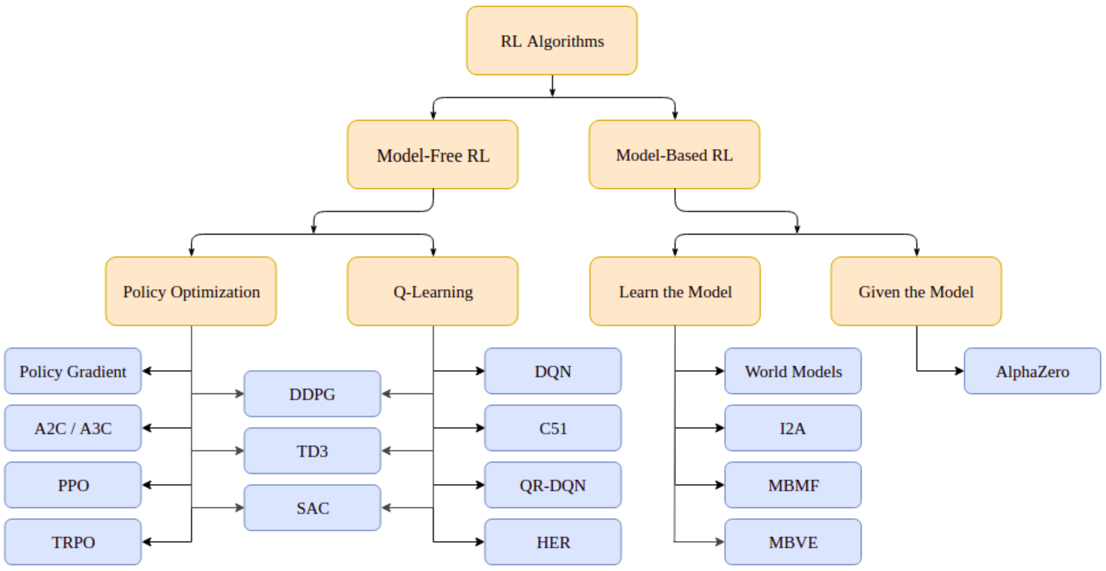


Figure 2.5. Taxonomy of algorithms in Modern RL [36]

For our objective of creating a dynamic search system that performs ranking using reinforcement learning, the literature survey was extremely crucial. With the help of the RLIRank paper, an end to end model that performs dynamic search was observed. With the survey on various ranking problems based on RL algorithms formulation of an MDP for our own setting was understood, and finally with the help of various research papers that shed light on the various algorithms of RL, experimentation and find results that have the potential to beat the state of the art scores were realised.

Figure 2.5 shows the generalized classification of the RL algorithms. Each of these algorithms are specific to the environment in question and the dataset used. The scope of the project and hence the thesis is limited to Model-Free Reinforcement learning algorithms. As mentioned earlier in literature review, Q-learning, DQN are both value based methods, policy gradient is a policy based method, A2C/A3C and PPO are actor-critic based methods. Each of these algorithms were further modified and experimented with as discussed in the further chapters.

2.6 Theoretical Discussions

Like any other machine learning algorithm, it can be observed that the tradeoff between variance and bias also persists in reinforcement learning algorithms. In general, an objective function J parameterised by θ can constitute to the making of the optimal policy which is provided by the following equation for policy gradient,

$$\nabla_\theta J(\theta) = E[\nabla_\theta log\ \pi_\theta(s_t, a_t)\ \psi_t] \tag{2.12}$$

where,

$\psi_t = R_t$, is the REINFORCE Algorithm

$\psi_t = R_t - b$, is the REINFORCE with baseline Algorithm

$\psi_t = Q^\pi(s_t, a_t)$,is the policy gradient theorem

$\psi_t = A^w(s_t, a_t)$, is the advantage actor critic Algorithm

$\psi_t = r_{t+1} + \gamma * V^w(s_{t+1}) - V^w(s_t)$,is the td actor critic Algorithm.

$\psi_t = \sum_{k=0}^{n-1} \gamma^k * r_{t+k+1} + \gamma^n * V^w(s_{t+n+1}) - V^w(s_t)$, is the n-step Actor critic Algorithm.

The more ψ_t relies on actual rewards like R_t the more the gradient will be correct on average i,e, have a small bias, but this implies that it will vary more i.e, have high variance. This increases the sample complexity, which requires averaging across more samples to accurately estimate the gradient. On the other hand, the more ψ_t relies on estimations like the TD error, the more stable the gradient will be i.e, have a small variance, but this implies it is more incorrect i.e, have high bias. This can lead to finding a suboptimal policy, i.e. local optima of the objective function. The n-step algorithm used in A2C and A3C is an attempt to balance these extremes. Schulman et al. proposed the Generalized Advantage Estimate (GAE) to further control the bias/variance trade-off in their proximal policy optimization[21].

2.7 Conclusions

While there exists some literature on the application of Reinforcement learning in information retrieval systems, the effectiveness and the efficiency of these algorithms are yet to be improved. With the literature survey done it can be seen that while ranking with reinforcement learning have baselines and dataset available publicly for comparison, a dynamic search model that takes user feedback into consideration and provides it to the RL agent for further ranking is still at its building stage. In spite of developing such a system there is a lack of methods available to perform verification of the entirety of the model and hence each module must be verified individually. With the extensive survey, a method that would serve the purpose of building such a system end to end can be concluded. Our objective is hence two-fold - firstly, we require to understand how the reinforcement learning model shall fit into a search system that takes user preference into account and secondly, how to improve on the learning to rank paradigm using reinforcement learning. Both of these objectives can be mapped with the help of the literature review done in this chapter.

CHAPTER 3

METHODOLOGY

3.1 Introduction

In this chapter, the methodology used to design and develop the dynamic search model with the Reinforcement Learning Component shall be discussed. Further the use of semantic models and the sliding window concept which is paramount in such a dynamic setting of information retrieval system is also discussed. Thereon, the chapter indulges the various algorithms used to experiment and potentially improve the results of the Reinforcement Learning agent. Thus, the algorithms used for the implementation and the design choices taken in order to fit them to the MDP formulation are discussed. Finally concluding with the various tools used and the implication of the experiments performed, the results of which are discussed in the next chapter.

3.2 Methodology

The methodology is structured into the following sections: Information Retrieval dynamic search system using RL, with a subsection devoted to each component implemented, implementation and experiments on Learning to Rank using Reinforcement Learning, with each subsection detailing the experimentation done to modify each algorithm.

3.2.1 Information Retrieval Dynamic Search System using RL

Initially at time T0, the system would fetch M documents from the corpus. Each of these documents would have a document ID and an initial relevance score of 1 defined along with a randomly initialized policy. These three terms together comprise the system's initial state.

Hence consider the state of the system (U_t) to be defined as:

$$U_t = ([D_t, Y_t], \pi_t) \tag{3.1}$$

where,

U_t = *State of the system at time t*
D_t = *List of Doc IDs at time t*
Y_t = *Corresponding relevance scores of the document IDs at time t*
π_t = *Policy of the RL agent at time t*

Relevance scores are the essence of the system. Our objective of the system would be to update the relevance scores accurately. This is because the relevance scores indicate the behaviour of the user feedback. If the relevance scores are learnt to accurately represent the documents that are relevant on the basis of the user feedback, it would then become independent of the query and would have a one-one correspondence with the documents remaining in the system's state.

This in contrast to similarity scores which are computed for each document with respect to a query, would not be able to generalize for the user's behavior of a document. For instance, a paper cited multiple times would have a higher user feedback and hence a higher relevance score than other less cited documents. However, the similarity score of the document is static and not changing for a query. Irrespective of the number of user feedbacks, a similarity score computed through a semantic model would generate the same result. Hence, updating relevance scores become the main objective of the system in order to capture the user's behaviour.

The system's state is then provided to the ranking module. The ranking module can also be interpreted as the Graphical User Interface of the simulator. This is where the user will be able to interact with the system and the system would be able to record the user's behavior in accordance to learn the same.

Consider that a query is fetched by a user. The ranking module would show a list of ranked documents to the user. Currently, the list of documents shown is set to ten. Based on the list obtained by the user, the user would manually record a user feedback which would be stored into one of the buffers of the ranking module. Once all the feedback has been recorded and stored, the system would then update its state by updating the relevance score and the parameters that would eventually update the policy of the RL agent. The update can be triggered either by the user itself, or by a set time period (an hour or a day or even a year). Once the time has elapsed the update shall happen and the system shall move to the next state with the updated relevance score and policy.

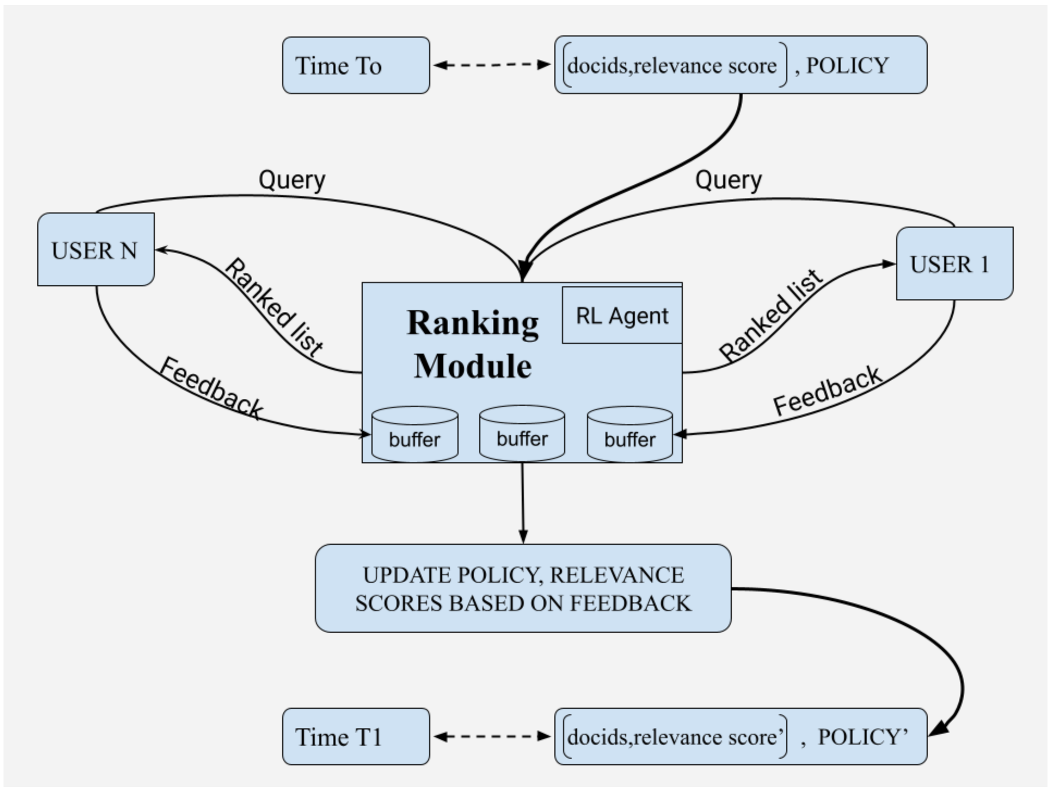


Figure 3.1 Overview of the IR system

3.2.1.1 Ranking Module

The task of the ranking module is to simply show the list of k documents to the user based on the query entered. Here, k is defined as the number of documents that is shown to the user based on which the user feedback will be taken.

As shown in Figure 3.1 the ranking module requires the system's state Ut as an input. Once the document IDs, and their corresponding relevance scores are given to the ranking module, it fetches the very first query. Using the paragraph and document embeddings from the trained doc2vec model, the numerical vector representation of the query is calculated. Consider that Ut has M number of documents. Each document in M is then mapped to its corresponding vector representation, and the cosine similarity score is computed with respect to the vector representation of the query. Note that the document ID is used to fetch the data of each document from the corpus (in the form of a document). Each of the document's cosine similarity calculated with respect to the query is defined as the similarity score of the document and query.

Based on the similarity score of each document with respect to the query , the documents and their corresponding relevance scores are sorted. Finally a retrieved list is produced by considering a linear weighted sum of similarity score and relevance score, i.e.

$$R = \alpha * SS + \beta * RS \tag{3.2}$$

where,

R = *Retrieved list of documents*

α = *constant (hyperparameter)*

β = *constant (hyperparameter)*

SS = *Similarity score of the documents*

RS = *Relevance score of the documents*

Consider that the list retrieved is for K documents. These are the number of documents that are shown to the user in order to record their user feedback. Based on the K documents retrieved the Reinforcement Learning agent would rank these documents on the basis of the policy π. Once the list of documents are ranked, the user would give the user feedback. The user feedback is to be incorporated into the system in order to be able to account for the user behaviour. In order to do this, the user feedback is used to generate the rewards for the RL agent. These rewards would essentially drive the RL agent, whose motive is to maximize these rewards. Based on the rewards the RL agent would then update the policy. To update the relevance scores, the user feedback is transformed to a numerical function and the relevance scores are also updated on the basis of the user feedback.

The updated relevance score, their respective documents and the updated policy of the agent is again provided to the ranking module. The entirety of the ranking module is executed whenever a query is fetched by the user or the simulator.

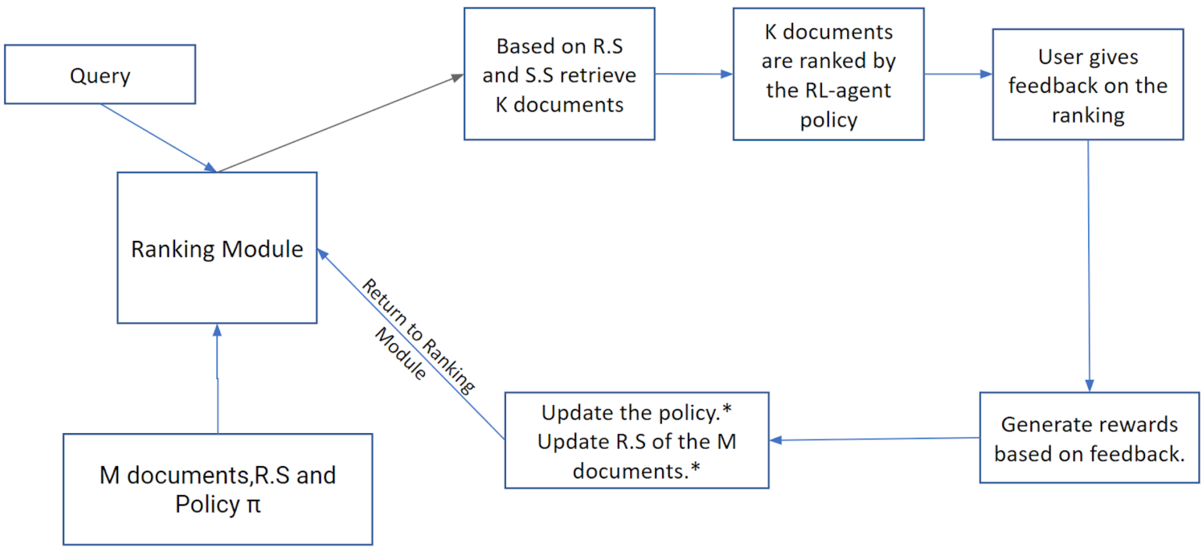

Query
Based on R.S and S.S retrieve K documents
K documents are ranked by the RL-agent policy
User gives feedback on the ranking
Ranking Module
Return to Ranking Module
Generate rewards based on feedback.
Update the policy.* Update R.S of the M documents.*
M documents,R.S and Policy π

Figure 3.2 Flowchart of the Ranking Module

3.2.1.2 Reinforcement Learning Agent

As mentioned earlier, the MDPRank setting fits our dynamic system. In MDPRank, the MDP is formulated as a tuple (S, A, T, R, π). Here,

S is a set of states each formulated as $s_t = [t, X_t]$,
R is a set of rewards each formulated as $r_t = R_{DCG}(s_{t-1}, a_{t-1})$,
A is a set of actions each formulated as $a_t \sim \pi(a_t|s_t;\ w)$,
T is the transition of a state given by $T(S, A)$ *i.e*, $T : S \text{ x } A \rightarrow S$,
Using T, the next state is updated as $s_{t+1} = T([t, X_t], a_t) = [t + 1, X_t \backslash \{x_{m(a_t}\}]$,
where,

s_t *= current state of the RL agent.*
t *= current timestep of the state*
X_t *= remaining documents in the state*
r_t *= rewards generated using DCG score*
a_t *= choosing a document for position t*
T *= Transition of state s_t to $s_t + 1$*

Here, the first and foremost question that appears is how must the documents be represented in order for it to be able to fit into the MDPRank setting. Since a Reinforcement Learning agent requires a feature vector representation of its state, we decided to use the feature vector of the document as done in the paper.

In order to fit this into our setting of queries and documents, both of which are provided in text (raw, unstructured data), we are required to convert these documents to their respective feature vectors. Fortunately, the semantic model used was Doc2Vec. Doc2Vec was used in order to generate the vector representation of the documents on the basis of the paragraph or document embeddings learnt during the training of the model.

With the feature vector of the documents obtained, we could now successfully fit our setting into the MDPRank setting. This also allowed for the analyses and verification of the MDPRank algorithm, its effectiveness and improvements of the algorithm, which would directly improve the system design.

The final formulation of the MDP after fitting to our model was then modified to,

S is a set of states each formulated $s_t = [t, X_t]$,
R is a set of rewards each formulated as $r_t = UserFeedback(X)$,
A is a set of actions each formulated as $a_t \sim \pi(a_t|s_t;\ w)$,
T is the transition of a state given by $T(S, A)$ *i.e*, $T : S \text{ x } A \rightarrow S$,
Using T, the next state is updated as $s_{t+1} = T([t, X_t], a_t) = [t + 1, X_t \backslash \{x_{m(a_t}\}]$,

where,

s_t = *current state of the RL agent.*

t = *current timestep of the state*

X_t = *remaining documents in the state*

r_t = *rewards generated using user feedback*

a_t = *choosing a document for position t*

T = *Transition of state* s_t *to* $s_t + 1$

Once the state is defined for the RL agent, the agent would then choose an action. The action is defined as choosing a document at timestep t. This chosen document would indicate the position of the document based on the policy π parameterised by *w.* The state is then transitioned by removing the document chosen by the agent from the list of documents. This would comprise an episode. The episode is defined as:

$$E = [((0,X_0),\ A_0),\ ((1,X_1),\ A_1),\ ((2,X_2),\ A_2),\ \dots ((M-1,X_{M-1}),\ A_{M-1})] \tag{3.3}$$

where,

M = length of the candidate set

All the actions hence chosen would result in the ranked list of documents. Hence the goal of the RL agent would be achieved which is to rank the documents. The ranking is said to be good when the learnt policy of the agent enables the agent to choose actions such that the rewards are maximised.

Atypical to the common RL problems, we had to choose to perform online updates. This essentially meant that the update was triggered as per the user or the elapsed time. The MDPRank formulation for the system, would return a ranked list of document based on policy π which is parameterised by the parameter w. The policy is defined as:

$$\pi(a_t|s_t; w) = \frac{exp\{w^T x_{m(a_t)}\}}{\Sigma_{a \in A(s_t)} exp\{w^T x_m(a)\}} \tag{3.4}$$

This is essentially the softmax on the feature vectors of the document chosen by the action *a* at time *t*. The agent hence would learn the optimal parameters *w* that would give the best policy π which would maximise the rewards *r.*

With this, the entire flow of the algorithm is as shown in Figure (3.3). One major distinction to note from that of the MDPRank algorithm is that once the action is completed and the agent's state is updated, the agent's policy is not updated immediately, since the rewards are not generated yet, which is triggered by the user feedback. Hence the need for a buffer that would hold all the user feedback for their respective queries.

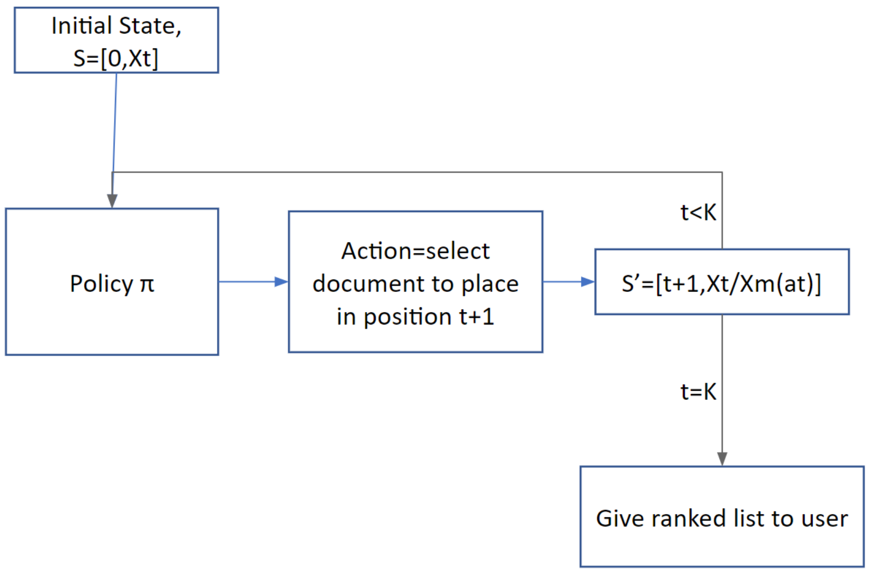


Figure 3.3. Flowchart of the Reinforcement learning module

The ranked list is now shown to the user, and the user feedback is recorded and stored in the buffers. Post completion of the user's query and feedback response, the user either clicks to update the system or the time would have elapsed with which the recorded user feedbacks would be used to generate the rewards. With the help of these rewards and the previously sampled episode E, the return and the score function is calculated, which is then used in the policy gradient algorithm of the MDPRank paper (described in Literature Review).

3.2.1.3 Sliding Window

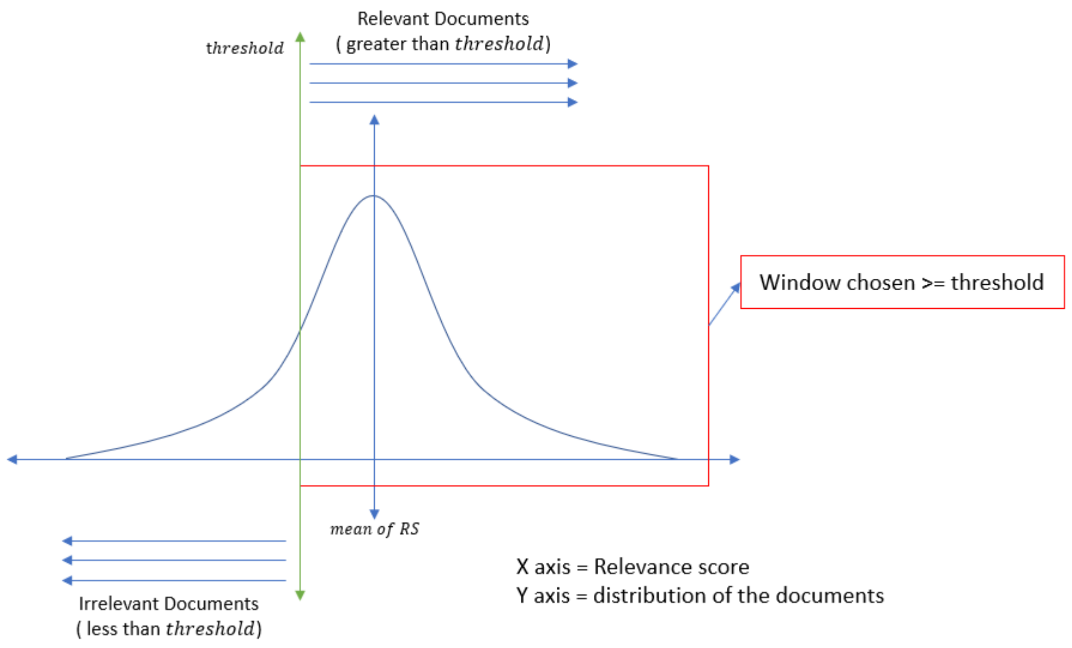


Figure 3.4 Concept of Sliding Window

In information retrieval systems, it is often the case that the size of the corpus is large. To be able to perform any kind of ranking on the entire corpus would be time consuming, considering that the results must be generated quickly. While the apt use of data structures could reduce the time, in the real-world scenario where data is a never ending stream, it is not possible to construct an algorithm that would be efficient irrespective of the amount of data. In order to tackle this problem a concept of sliding window was introduced into the system.

As the number of documents added to the system keeps increasing, some documents would lose their relevance. Irrespective of the query, there may be documents that are not relevant and the user never gives a positive feedback to. The sliding window allows for more documents to be added while removing documents. As shown in fig (above), as the relevance score of the documents are updated by the system a threshold is initiated in order to remove the documents that have relevance scores less than or equal to 0. In order to ensure that there would be documents that get removed, the documents are introduced with a linear decay when there is no user feedback given to it. This ensures that eventually, after sufficient updates and addition of documents the sliding window would remove documents from the system. Due to this, the inefficiency of the system when handling more documents is stabilised, and hence reduces the loss of speed of retrieval and ranking irrespective of the amount of data incoming.

3.2.1.4 User Feedback

In a system such as this, user feedback plays an extremely important role. The more accurate the user feedback the better the system shall perform. However, due to the size of the dataset, it is not possible to be able to record users that would provide feedback on each query document pair. In order to tackle this, a semantic model like Doc2Vec is used, to get the cosine similarity score of the top 2 - 5 lines of the document with respect to the query. This is done in order to mimic an actual user who would click on a link reading the first 2 lines of the description as provided by an information retrieval system like a search engine.

3.2.1.5 Complete High Level IR System constructed

With the help of the modules discussed above, a complete system was hence formulated, that made use of the following features:

1. Concept of a relevance score that was independent of the query or rather denoted how good the documents were across the queries.
2. A ranking module that takes into consideration both the similarity of the documents to the query and also the behaviour of the user.
3. An Reinforcement learning agent that learns to rank the documents on the basis of the user behaviour that is inbuilt as a part of the ranking module.

4. The ability of the users to fetch queries and provide feedback that would serve as rewards for the RL agent.

5. The ability of the system to generate user feedback with the help of a semantic model and/or record feedback from users themselves.

All these features were then mapped to an end to end information retrieval system as shown in Figure 3.5. The system starts with initial M documents and the GUI interface (or the simulator) is called. The very first query is fetched or generated and based on the query, similarity scores are calculated for the M documents. Using a linear combination of the similarity and the relevance scores, K documents are retrieved and given to the RL agent. Based on the initialized policy, the RL agent ranks the K documents and this is then shown to the user. The user either generates or provides user feedback which are all collected and stored in a dictionary of dictionaries for O(1) access. Thereon, either the time t is made to elapse or the user triggers the update of the system. During the phase of updation, the user feedbacks are used to generate rewards for the RL agent system, with which the RL agent learns or updates to a better policy. With the help of sliding window concept documents are added and the documents which have lost relevance due to linear decay and lack of frequent positive feedback from the user(s), are evaluated and removed if the relevance score of the document is less than or equal to 0. This new list of documents along with the updated corresponding relevance scores and policy is considered as a new state of the system (U_{t+1}). Hence the system would have transitioned to the next state.

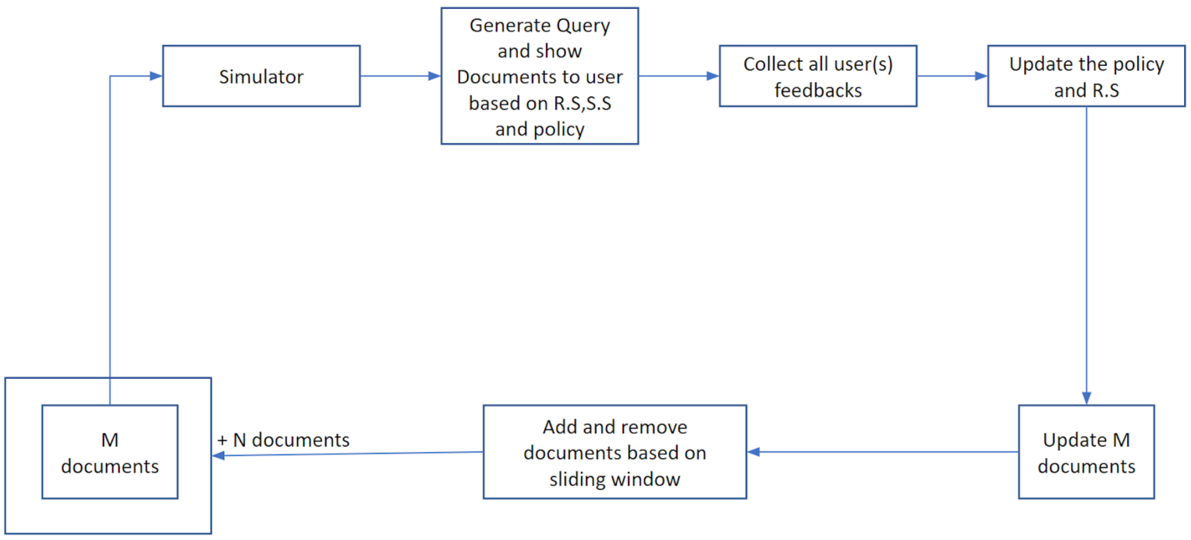

Simulator
Generate Query and show Documents to user based on R.S,S.S and policy
Collect all user(s) feedbacks
Update the policy and R.S
Update M documents
Add and remove documents based on sliding window
+ N documents
M documents

Figure 3.5 High Level Flowchart of the Information Retrieval System for Dynamic Search

3.2.2 Implementation and Experiments using RL models to perform Ranking

In this section we discuss the various implementations and experiments performed in order to improve on the scores obtained by the Learning to Rank task performed using Reinforcement Learning on the MQ2007 and MQ2008 datasets.

3.2.2.1 Implementing MDPRank Paper [13]

The RL agent as mentioned above was based upon the MDPRank paper. The reason for using this paper is because of the well structured Markov Decision Process formulation done in the paper for the ranking problem. It is also highly cited by other research papers, and provides an RL ranking baseline that beats previous ranking algorithms achieved by non-reinforcement learning machine learning and deep learning methods. This paper gave a direction in which we could use an MDP formulation for the ranking problem. The RL agent must hence be thought of as a separate standalone problem, without the retrieval system. Our task for the RL agent was broken into implementing the MDPRank paper's algorithm on the dataset used by the paper and reproducing the results. Once this was achieved, we moved on to ways in which the RL agent could be improved. To reproduce the results, we followed the MDPRank paper's algorithm as given by algorithm (3.1.1) and algorithm (3.1.2) [13].

Algorithm 3.1.1 MDPRank Learning

```
INPUT: Training set D = $\{(Q^{(n)}, X^n, Y^n)\}$ for n = 1 to N, learning rate $\eta$, discount factor $\gamma$, and
reward function R
OUTPUT: w
1.  Initialize w ← random values
2.  repeat
3.      $\Delta w = 0$
4.      for all (q, X, Y) ∈ D do
5.          $(s_0, a_0, r_1, \dots s_{M-1}, a_{M-1}, r_M) \leftarrow SampleAnEpisode(w, q, X, Y, R)$
6.          for t=0 to t=M-1 do
7.              $G_t \leftarrow \sum_{k=1}^{M-t} \gamma^{k-1} r_{t+k}$
8.              $\Delta w \leftarrow \Delta w + \gamma^t G_t \nabla_w \log \pi(a_t|s_t; w)$
9.          end for
10.     end for
11.     $w \leftarrow w + \eta \Delta w$
12. until converge
13. return w
```

Algorithm 3.1.2 $SampleAnEpisode$

```
INPUT: Parameters w, q, X, Y and R
OUTPUT: An episode
1. Initialize $s_0 \leftarrow [0, X], M \leftarrow |X|, and\ episode\ E \leftarrow \emptyset$
2. for t=0 to t=M-1 do
3.     Sample an action $a_t \in A(s_t) \sim \pi(a_t|s_t; w)$
4.     $r_{t+1} \leftarrow R(s_t, a_t)$ $\{Calculation\ on\ the\ basis\ of\ Y\}$
5.     Append $(s_t\ a_t, r_{t+1})$ at the end of E
6.     State transition $s_{\{t+1\}} \leftarrow [t+1,\ X \setminus \{x_{m(a_t)}\}]$
7. end for
8. return E = $(s_0, a_0, r_1, \dots s_{M-1}, a_{M-1}, r_M)$
```

The algorithm makes use of a policy based reinforcement learning technique called the policy gradient method (REINFORCE algorithm). The notations of the algorithm are as discussed earlier in section 3.2.1.2. The implication of the rewards were however different. In this algorithm the rewards were generated in order to directly correlate to the evaluation metric, that is the DCG (discounted cumulative gain score). Hence the reward defined by the MDPRank paper is,

$$R_{DCG}(s_t,\ a_t) \ = \ \sum_{i=1}^{t} \frac{2^{rel_i}}{log_2(i+1)} \qquad (3.5)$$

Another term introduced in the paper which is specific to the problem is the relevance label denoted as Y. This indicates the relevance of the document (X) with respect to the query (Q). Y denotes discrete values, which are subsequently linked to irrelevant (0), partially relevant (1) and completely relevant (2). These are however dependent on the datasets and may have a smaller or larger range of discrete values. The algorithm's goal is to maximise the expected long term returns which is calculated for each position of the document. To verify the results of the algorithm, the MQ2007 and MQ2008 [32] dataset was used.

The implementation when followed gave subpar results to that mentioned in the paper. This was noticed in other implementations of the paper too. One of the implementations sampled the best possible action by considering a supervised ordered set of relevance label. This when introduced to our implementation gave comparable results to the results stated in the MDPRank paper (discussed further in Chapter 4).

3.2.2.2 Implementing PPGRank Paper

As mentioned in the literature review, a natural extension of the MDPRank paper was given by the PPGRank paper. This algorithm tackled the issues of the MDPRank paper (discussed in Chapter 2) by using the pairwise learning to rank approach. Pairwise Policy Gradient (PPG) aims to sample 2 episodes both starting from the same state. The states are then updated on the basis of the scores and gradient directions and this is further used to update the parameters of the MDP. The paper further proves that this approach is theoretically sound as it is a “natural generalization of REINFORCE, and it can be considered as a variation of REINFORCE with baseline”[17]. The algorithm is as shown in 3.2.

While the algorithm shares the same notations as that of MDPRank paper, it samples 2 episodes for every state unlike its former half. MDPRank paper requires to sample the entire trajectory only once for each query. Due to this, when extending the MDPRank implementation to PPGRank implementation, the algorithm had the time complexity O(EQM(M+1)) where E is the number of epochs, Q is the number of queries and M is the average number of documents per query. This considerably increased the time when implementing the sequential version of the algorithm, however it converged faster than the MDPRank paper to the results described by the PPGRank paper without the need of sampling the best action. This indicated that our previous implementation of the MDPRank paper was correct and that further experiments could be done taking that as the base implementation.

Algorithm 3.2 Pairwise Policy Gradient (PPG)

```
INPUT: Training set D = ${(Q^{(n)}, X^n, Y^n)}$ for n = 1 to N, learning rate $\eta$, discount factor $\gamma$, and
reward function R
OUTPUT: $\theta$
1.  Initialize $\theta \leftarrow$ random values
2.  repeat
3.      $\Delta\theta = 0$
4.      for all (q, X, Y) $\in$ D do
5.          Initial state $S = S(q, X) \{Init\ state\ with\ Q\}$
6.          for t=0 to t=M-1 do
7.                  $\tau^A = \{S, A_T, R^A_{t+1}, S^A_{t+1}, \dots, S^A_{M-1}, A_{M-1}, R^A_M\} \sim \pi(\theta)$
8.                  $G^A \leftarrow \sum_{k=1}^{M-t} \gamma^{k-1} R^A_{t+k} \{long\ term\ return\ of\ \tau^A\}$
9.                  $\tau^B = \{S, B_T, R^B_{t+1}, S^B_{t+1}, \dots, S^B_{M-1}, B_{M-1}, R^B_M\} \sim \pi(\theta)$
10.                 $G^B \leftarrow \sum_{k=1}^{M-t} \gamma^{k-1} R^B_{t+k} \{long\ term\ return\ of\ \tau^B\}$
11.                 $\Delta\theta \leftarrow \Delta\theta + (G^A - G^B) . ( (\nabla \log \pi(A_t|S_t; \theta) - (\nabla \log \pi(B_t|S_t; \theta))$
12.                 $S \leftarrow \begin{cases} S^A_{t+1} & G^A \geq G^B \\ S^B_{t+1} & otherwise \end{cases}$
13.         end for
14.     end for
15.     $\theta \leftarrow \theta + \eta\Delta\theta$
16. until converge
17. return $\theta$
```

3.2.2.3 Experiments with Actor Critic Algorithms

Actor Critic algorithms make use of the best of value based methods and policy based methods. While value based methods estimate the optimal value function that maps action to value, policy based methods find the optimal policy without any Q-value estimated[35]. Actor critic methods basically, try to merge both these concepts. The actor is used to sample an action based on a state and the critic produces the Q values of the action. The actor hence tries to learn the optimal policy while the critic evaluates the learnt policy by calculating the value function of the action chosen by the actor. Hence enabling both of them to learn simultaneously.

In the case of the MDP formulated by the MDPRank setting, a subtle difference is noticed from the typical environments used with actor-critic methods. The state space of the MDP changes with every iteration. While this does not impose a problem for the actor, it creates a difficulty in calculating the value function.

3.2.2.3.1 Q-Actor Critic Algorithm (QACRank)

Our first successful approach to estimating a value function that provided learning in the actor-critic setting, was the Q-actor critic algorithm. This algorithm makes use of the policy gradient theorem, which, defines policy gradient as,

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta log\, \pi_\theta(s,a)\, Q^{\pi_\theta}(s,a)] \qquad (3.6)$$

where,

J = any of the policy objective functions,

$\pi_\theta(s,a) = any\ differentiable\ policy$

To reduce variance, we further use a critic to estimate the action-value function ($Q^{\pi_\theta}(s,a)$),

$$Q_w(s,a) \approx Q^{\pi_\theta}(s,a) \qquad (3.7)$$

Note that this holds if the approximator with parameter w is compatible with the policy. Therefore, the actor-critic algorithms have 2 parameters, one for actor that updates the policy parameters (θ), as directed by the critic and another for the critic that updates the action-value function parameters (w). The pseudo code of the methodology followed for the implementation is as shown in the algorithm below.

Algorithm 3.3 QACRank

INPUT: Training set D = $\{(Q^{(n)}, X^n, Y^n)\}$ for n = 1 to N, actor's learning rate α, critic's learning rate β, discount factor γ, and reward function R

OUTPUT: w, θ

```
1.  Initialize w ← random values; Initialize theta ← with random values
2.  repeat
3.      for all (q, X, Y) ∈ D do
4.          Initial state $S = S(q,X)\{Init\ state\ with\ Q\}; M = |X|$
5.          for t=0 to t=M-1 do
6.                  Sample an action $a_t \in A(s_t) \sim \pi(a_t|s_t;\theta)$ {Equation 3.8}
7.                  $r_{t+1} \leftarrow R(s_t, a_t)\ \{Calculation\ on\ the\ basis\ of\ Y\}$
8.                  $s_{t+1} \leftarrow [t+1, X \setminus \{x_{m(a_t)}\}]$
9.                  $\delta_t = r_{t+1} + \gamma * \hat{Q}_w(s_{t+1}, a_{t+1}) - Q_w(s_t, a_t)$ {Equation 3.9}
10.                 $w_{t+1} = w_t + \beta\delta_t\nabla_w Q_w(s,a)$
11.                 $\theta_{t+1} = \theta_t + \alpha Q_w(s,a)\nabla_\theta \log\pi(s_t, a_t;\theta)$
12.                 Transition $S \leftarrow S_{t+1}$
13.         end for
14.     end for
15. until converge
16. return w, $\theta$
```

In this algorithm, initially a state S is defined which starts the learning for the set of documents related to the first query. The state S is stored in order to calculate the Q-value of the state parameterised by w later. The updates in this algorithm are made to happen incrementally or stepwise. So the first objective is to sample an action based on the policy π parameterised by θ as given by equation 3.8

$$\pi(a_t|s_t;\theta) = \frac{\exp\{\theta^T x_{m(a_t)}\}}{\sum_{a \in A(s_t)} \exp\{\theta^T x_{m(a)}\}} \qquad (3.8)$$

Similar to the MDPRank algorithm, the rewards and the next state, which is the state of remaining documents in the list post-removal of the document based on which the action is taken, is then calculated. The state is not transitioned to the new state yet, but instead, the new state is only stored. The Q-value of the state is calculated and the Q-value of the next state is then approximated both with the help of equation 3.9

$$\hat{Q}_w(s_{t+1}, a_{t+1}) = w^T\,\psi(s_t, a_t) = \nabla_\theta \log\pi_\theta(s_t, a_t) \qquad (3.9)$$

With the values computed, the updates for the critic's parameter (w) and the actor's parameter (θ) are done. Once the updates are completed, the transition of the state is done to the next state. Note that, there does not exist an approximated Q-value of the next state, at the last state. For this the approximated Q-value for the next state is taken as 0.

3.2.2.3.2 Advantage Actor Critic

As explained in Chapter 2 the policy gradient theorem can be written with the value function, which is a more optimal baseline than the Q-value. But for our setting, the issue is the calculation of the value function. In order to be able to calculate the value function of the state, we had to ensure that we would obtain $\Phi(s)$ which is the feature vector representation of state S. Thus, We need to compute,

$$\hat{V}(s_t) = \Phi(s_t)^T w_t \tag{3.10}$$

where,

$\Phi(s)$ which is the feature vector representation of state S

w is the weight vector.

To compute $\Phi(s_t)$, we considered the already present $\Phi(s, a)$ which is the feature vector of the document chosen by the agent. Thus we can conclude,

$$\Phi(s) = E_{\pi(a,\theta)}[\Phi(s, a)] \tag{3.11}$$

which gives us,

$$\hat{V}(s, w) = E_{\pi(a,\theta)}[q(s, a)] = \Phi(s_t)^T w_t \tag{3.12}$$

Hence, allowing us to use either a linear or a nonlinear function approximator to estimate the value function.

Algorithm 3.4 Advantage Actor Critic with TD(0) updates

INPUT: Training set D = $\{(Q^{(n)}, X^n, Y^n)\}$ for n = 1 to N, actor's learning rate α, critic's learning rate β, discount factor γ, and reward function R

OUTPUT: w, θ

```
1.  Initialize w ← random values; Initialize theta ← with random values
2.  repeat
3.      for all (q, X, Y) ∈ D do
4.          Initial state $S = S(q, X)\{Init\ state\ with\ Q\}; M = |X|$
5.          for t=0 to t=M-1 do
6.              Sample an action $a_t \in A(s_t) \sim \pi(a_t|s_t; \theta)$
7.              $r_{t+1} \leftarrow R(s_t, a_t)$ $\{Calculation\ on\ the\ basis\ of\ Y\}$
8.              $s_{t+1} \leftarrow [t+1, X \setminus \{x_{m(a_t)}\}]$
9.              $\delta_t = r_{t+1} + \gamma * \hat{V}_w(s_{t+1}) - \hat{V}_w(s_t)$ {Computed with Equation 3.10}
10.             $w_{t+1} = w_t + \beta\delta_t\nabla_w \hat{V}_w(s_t)$
11.             $\theta_{t+1} = \theta_t + \alpha\delta_t\nabla_\theta \log\pi(s_t, a_t; \theta)$
12.             Transition $S \leftarrow S_{t+1}$
13.         end for
14.     end for
15. until converge
16. return w, $\theta$
```

Algorithm 3.5 Advantage Actor Critic with Monte-Carlo updates

```
INPUT: Training set D = $\{(Q^{(n)}, X^n, Y^n)\}$ for n = 1 to N, actor's learning rate $\alpha$, critic's learning rate $\beta$, discount factor $\gamma$, and reward function R, episode length T
OUTPUT: w, $\theta$
1.  Initialize w ← random values; Initialize theta ← with random values
2.  repeat
3.      for all (q, X, Y) ∈ D do
4.          Initial state $S = S(q, X)\{Init\ state\ with\ Q\}$;
5.          for t=0 to T do
6.              Sample an action $a_t \in A(s_t) \sim \pi(a_t|s_t; \theta)$ {Equation 3.8}
7.              $r_{t+1} \leftarrow R(s_t, a_t)\ \{Calculation\ on\ the\ basis\ of\ Y\}$
8.              {Store $(s_t, a_t, r_{t+1})$ }
9.              $s_{t+1} \leftarrow [t+1, X \setminus \{x_{m(a_t)}\}]$
10.         end for
11.         {Compute all T step returns}
            $G_t = \sum_{k=1}^{T-t} \gamma^{k-1} r_{t+k}$
12.         for t = 0 to T do
                $A_t = (G_t - \hat{V}_w(s_t))$
13.             $\theta_{t+1} = \theta_t + \alpha A_t \nabla_\theta \log \pi(s_t, a_t; \theta)$
                $w_{t+1} = w_t + \beta A_t \nabla_w \hat{V}_w(s_t)$
14.         end for
        end for
15. until converge
16. return w, $\theta$
```

Based on various implementations available on actor critic methods, there are 2 ways in which the advantage actor critic can be used in the setting.

1. Calculating the advantage function as the TD error, and performing updates stepwise.
2. Calculating the advantage function with the help of true rewards and performing updates at the end of the episode.

The latter half is more celebrated in terms of the number of implementations available online and the number of environments solved by the agent. More environment solved ensures the robustness of the algorithm and leads to more promise. However, for the sake of experimentation and theory soundness, both these methods were tested for our setting. Algorithm 3.4 denotes stepwise updates with the help of TD updates and Algorithm 3.5 shows updates at the end performed by Monte Carlo Updates.

3.2.2.3.3 Proximal Policy Optimization with Monte Carlo Advantage Estimates

The proximal policy optimization algorithm (PPO) is a well received algorithm in the literature. The algorithm's main objective is to allow for an implementation of trust region update while keeping it compatible with stochastic gradient descent. This allows for a better control of the policy changes at each iteration. The objective function used is given by [18] :

$$L_t^{CLIP}(\theta) = E_t[\min(r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t) \qquad (3.13)$$

where,

θ is the policy parameter

E_t denotes the empirical expectation over timesteps

r_t is the ratio of the probability under the new and old policies

A_t is the estimated advantage at time t

ϵ is hyperparameter (usually 0.1 or 0.2)

While the official algorithm (algorithm 2.1) makes use of parallel actors, it is important to note that it does not lead to any computational speed improvement. The use of parallel actors have been done in order to improve the exploration done by the algorithm. Instead of having a single actor start from the initial state, multiple actors are made to start from the same initial state and based on the different trajectories, the policy is given the ability to learn a better update. Therefore in our setting, the parallel version of PPO would not decrease time taken to run the algorithm, but could enable learning to a better optimum. But, there was one major issue that was faced while trying to implement Parallel PPO. The updates in PPO occurred across the queries. In order to parallelize this, it is required that we must share the state space to each actor. However, in our setting the state space is changing due to which it creates a jagged array. Such an array cannot be passed between the actors with the help of the parallel libraries available in python, since they deal with a fixed state space. Hence instead of implementing a parallel version of PPO it was preferable to implement it's sequential version. The algorithm hence followed for our setting is as shown in Algorithm 3.6.

Algorithm 3.6 PPO Algorithm using Monte Carlo Advantage Estimate

INPUT: Training set D = $\{(Q^{(n)}, X^n, Y^n)\}$ for n = 1 to N, actor's learning rate α, critic's learning rate β, discount factor γ, and reward function R, episode length T, epochs K, epsilon clip ϵ, minibatch M, value function coefficient c1, entropy coefficient c2.

OUTPUT: w, θ

```
1.  Initialize w ← random values; Initialize theta ← with random values
2.  for iteration 1,2 … do
3.      for all (q, X, Y) ∈ D do
4.          Initial state $S = S(q, X)\{Init\ state\ with\ Q\}$;
5.          for t=0 to T do
6.              Sample an action $a_t \in A(s_t) \sim \pi(a_t|s_t; \theta)$ {Equation 4.1}
7.              $r_{t+1} \leftarrow R(s_t, a_t)$ $\{Calculation\ on\ the\ basis\ of\ Y\}$
8.              {Store $(s_t, a_t, r_{t+1})$ }
9.              $s_{t+1} \leftarrow [t + 1, X \setminus \{x_{m(a_t)}\}]$
10.             $s_t \leftarrow s_{t+1}$
11.         end for
12.         {Compute T step advantage estimates}
            $A_t = G_t - V(s_t)$
                where,
                $G_t = r_t + \gamma * r_{t+1} + \gamma^2 * r_{t+2} + \cdots + \gamma^{T-t-1} * r_{T-1} + \gamma^{T-1} * V(s_T)$
13.         {Optimize for K epochs}
14.         for epoch 1,2,.. K do
15.             {Sample minibatches of size M where M<=T}
16.             for each minibatch do
17.                     $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$
18.                     $L_t^{CLIP}(\theta) = E_t[\min(r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)$
19.                     $L_t^{VF}(w) = (V_w(s_t) - V_t^{targ})^2$
20.                     $L_t^S(\theta) = S[\pi_\theta](s_t)$
21.                     $L_t = E(L_t^{CLIP}(\theta) - c_1 * L_t^{VF}(w) + c_2 * L_t^S)$
22.                     $\{optimize\ using\ adam\ optimizer\ on\ the\ basis\ of\ L_t\}$
23.             {Clear all buffers}
24.             end for
25.         end for
26.     end for
27. end for
28. return w, $\theta$
```

3.2.2.4 Deep Q Network

An experiment was also performed using a value based iteration method called the Deep Q network as reviewed in chapter 2. As mentioned earlier, to solve the issues caused by Q-learning methods, an experience replay memory (or simply a buffer) is introduced, which stores these minibatch transitions. Using stochastic gradient descent on the experience replay memory, it randomly samples minibatches and hence trains the network. Once the buffer has reached its full capacity, the new transitions shall replace the old transitions. To tackle the issue of non-stationarity, the targets are computed with respect to the old parameters of the neural network and then updated in small time intervals with the new learned parameters. The algorithm used is as shown below.

Algorithm 3.7 Deep Q Network Algorithm (DQNRank)

```
INPUT: Training set D = {($Q^{(n)}, X^n, Y^n$)} for n = 1 to N, learning rate $\alpha$, discount factor $\gamma$, and
reward function R, episode length T, epsilon clip $\epsilon$, minibatch M.
1.  Initialize D ← Replay memory; Initialize parameter w ← with random values; Initialize
    parameter v that calculates TD-Target by v ←w
2.  for epoch 1,2 … N do
3.      for all (q, X, Y) ∈ D do
4.          Initial state $S = S(q, X)\{Init\ state\ with\ Q\}$;
5.          for t=0 to T do
6.                  Sample a random action with probability $\epsilon/m$ otherwise
                    select action $a_t = argmax_a\ \hat{q}(s, a, w^t)$
7.                  $r_{t+1} \leftarrow R(s_t, a_t)\ \{Calculation\ on\ the\ basis\ of\ Y\}$
8.                  $s_{t+1} \leftarrow [t+1, X \setminus \{x_{m(a_t)}\}]$
9.                  {store $(s_t, a_t, r_{t+1}, s_{t+1})$ tuple in D }
10.         end for
11.         Sample random batch from D
12.         $g \leftarrow r_t + \gamma \max_a \hat{q}(s_{t+1}, a', v)$
13.         $\hat{g} \leftarrow \hat{q}(s, a, w^t)$
14.         $w^{t+1} \leftarrow w^t - \alpha * 0.5 * \nabla(g - \hat{g})^2$
15.         Every c time steps, set $v \leftarrow w^t$
16.     end for
17. end for
```

3.3 Tools used

We use Python as our main programming language and for the implementation of neural networks we use Pytorch. We make use of Tkinter api for the GUI interface of the simulator of the IR Ranking system. We further use numpy as the library for matrix computations. For the implementation of Doc2Vec model we also made use of the Gensim Library implementation. We also extensively made use of google Colab for processing parallel libraries as done in the implementation of PPGRank.

3.4 Preliminary Analysis

With the help of the rewards graph, an early analysis could be performed. It could be observed that the algorithms were extremely sensitive to their hyperparameters. A slight change in the magnitude order of 1e-05 could lead to drastic improvements or fall in the trend of the results. This is because of the number of updates that were happening instantaneously for the problem description. Since for every document a corresponding update is happening, and in some cases after every 1000 or 2000 steps, the learning of the agent is high. This is necessary as Reinforcement Learning algorithms are notoriously known to be highly time taking and taxing as compared to the other machine learning algorithms. It requires more runs than the other machine learning techniques and more analysis that needs to be performed at every stage. The requirement of unit testing every component is hence necessary too. Taking all these practices into consideration, the experimentation for these algorithms on the dataset took time and eventually led to the results that were expected and in some cases, beating the state-of-the-art results. Among the many algorithms, it was also observed that the proximal policy optimization was less sensitive to hyper-parameters as compared to the other algorithms. This is due to the surrogate loss performed to the objective function.

3.5 Conclusion

To conclude, the algorithms provided in research papers were used and implemented to gain a better understanding of the implementation of Reinforcement learning problems. Further a simulator was created from scratch that used the Information Retrieval system design as discussed in this chapter. This simulator allowed the capture of user feedback for datasets which otherwise did not have any. The user feedback is essential for the project as it defines the problem statement to be learning “dynamically”. Further, experimentation with different algorithms for the Reinforcement learning ranking module were performed that gave insights on whether the scores could be improved and whether given a user feedback our agent would learn or not. The results of each of the above algorithms mentioned in this chapter is discussed in Chapter 4.

CHAPTER 4

RESULT ANALYSIS

4.1 Introduction

This section aims to give an overview of the best results obtained. Like mentioned earlier, the performance of the system is in direct correlation to the performance of the Reinforcement learning agent. This section will hence describe the results obtained by each of the algorithms fit into the Markov Decision Process formulation. We then proceed to analyse and compare these results with the state of the art scores and the baselines provided for the MQ2007 and MQ2008 dataset. Further, the system's final graphical user interface along with the command line prints are shown and explained. Thereafter, the chapter is concluded with the interpretation of the results and summarisation of the results obtained.

4.2 Result Analysis

To evaluate the results and to form a fair comparison, the NDCG (normalized discounted cumulative gain) scores are considered for each model. The NDCG score at the Kth position indicates the number of documents ranked correctly upto the Kth position with respect to the perfect ranking of the documents upto the Kth position. The results generated for the dataset are with respect to a 70-30 split. 70% of the dataset is considered as the training set and 30% of the dataset is considered as the testing set. These results indicated a similarity to those generated with 5 Fold cross validation. Only the results of the MDPRank and PPGRank paper were produced across the 5 folds; the rest was produced on the entire dataset with 70-30 split.

4.2.1 MDPRank

Essentially, the results of the MDPRank paper would verify the paper and the formulation provided in the paper. After trying with various hyperparameters and heuristics we could reproduce the results with a ± 0.02 to 0.04 variance. We noticed that sampling the action based on a random policy would consistently give subpar results, and on further inspection of one of the author's code on Github, the results were being produced by sampling the best action wherein the action chosen by the policy is based upon the document with the highest relevance label. Upon using this heuristic we were able to achieve the results as shown in table 4.1. The results were achieved by running the algorithm on MQ2007 for 50 epochs and MQ2008 for 150 epochs with a learning rate of 0.05 and setting gamma to 1.

Table 4.1 Reproduced MDPRank NDCG Scores

K	MQ2007	MQ2008
1	0.3954	0.3741
3	0.3983	0.4099
5	0.4069	0.4569
10	0.4350	0.2230

4.2.2 PPGRank

Table 4.2 Reproduced PPGRank NDCG scores

K	MQ2007	MQ2008
1	0.3947	0.3741
3	0.3993	0.4226
5	0.4089	0.4627
10	0.4382	0.2290

The PPGRank implementation results proved that the underlying code of MDPRank was robust and correct as without any heuristics PPGRank implementation which is an extension of the MDPRank implementation provided the results of the paper with a ± 0.02 to 0.04 variance. It can be noticed that the results provided by the PPGRank implementation were much closer than that provided by the MDPRank implementation. The scores were generated by setting the learning rate as 0.05 and sampling action based on categorical distribution. It was also noticed that the sequential implementation of PPGRank paper would take more time to complete each epoch, but the number of epochs needed for it to converge to the values was less. This was in accordance with the low variance property of the pairwise approach as mentioned in the paper [17]. The results are as shown in table 4.2

4.2.3 Q- Actor Critic Algorithm

The Q-Actor Critic algorithm introduces our first successful experiment to fit the actor-critic algorithm to the MDP formulation. This implementation used a linear function approximator. That is, it did not make use of neural networks. Performing step updates with the actor's learning rate fixed to 0.0001, critic's learning rate fixed to 0.0002 and gamma fixed to 1, the results shown in table 4.3 were obtained running for 50 epochs. By performing various hyperparameter tuning it was concluded that a stable increase in learning is seen for the testing set only when small learning rates for actor and critic is used. The scores obtained at the end of the 50th epoch are as shown here. Note that to achieve these scores, like MDPRank implementation, sampling the best action at every timestep is done. However, when applying a pairwise approach to the same, no sampling of best action were required and similar scores could be achieved.

Table 4.3 NDCG Scores of Q-Actor Critic Algorithm (QACRank)

K	MQ2007	MQ2008
1	0.3694	0.3658
3	0.3759	0.3920
5	0.3827	0.4292
10	0.4120	0.2053

4.2.4 Deep Q Network

The Deep Q Network implementation of the MDPRank formulation experiments a value based iteration method. Tuning the hyperparameters and running it for 100 epochs using 5e-05 learning rate, gamma of 1.0, replace target as 2000, with epsilon start, end and decay set to 1, 0.03 and 0.005 respectively along with a memory batch size of 128 and having a maximum memory size of 50000. It can be inferred from experiments Having a large memory size as well as mini-batch size in proportion to the max episode length allows the algorithm to pick random transitions across different queries which reduces the chances of overfitting. The neural network uses RELU activation function with a single hidden layer having 32 hidden nodes. The results are as shown in table 4.4

Table 4.4 NDCG Score for DQN (DQNRank)

K	MQ2007	MQ2008
1	0.4022	0.3376
3	0.4041	0.3887
5	0.4069	0.4313
10	0.4254	0.2320

4.2.5 Advantage Actor Critic

Most implementations of actor-critic given online with respect to other environments perform advantage actor-critic. As mentioned previously in Chapter 3, the updates for such algorithms could be done either stepwise or batchwise. We consider 2 experimental updates, one that performs a stepwise approach in accordance with algorithm 3.4 and another that performs updates at the end of the trajectory as in algorithm 3.5. The stepwise approach failed to learn with a linear function approximator; however, by sampling the best action this approach provided a learning curve for the testing values, but the scores, although comparable, were subpar to the expected results. Upon implementing the same with neural networks (non-linear function approximator) using separate actor and critic networks, better and comparable scores were achieved. The actor network used a single hidden layer with (64,32) dimensions along with TanH activation function whereas the critic network used a single hidden layer with (64,32) dimensions along with RELU activation function. The hyperparameters used were different for the MQ2007 and MQ2008 dataset. Both results obtained were without sampling the best action. The hyperparameters used for the actor's learning rate was 1e-05, critic's learning rate was 2e-05, gamma was fixed to 1.0 and the episode length was fixed to 20. Both were run for 100 epochs. The results are as shown in Table 4.5

Table 4.5 NDCG Score for Advantage Actor Critic TD(0) (IncrRank)

K	MQ2007	MQ2008
1	0.3786	0.3969
3	0.3871	0.4148
5	0.3953	0.4744
10	0.4198	0.2533

On performing the same experiments with advantage actor-critic performing updates at the end of the trajectory, by collecting all rewards for the entire length of the episode, using actor learning rate as 0.0001 and critic's learning rate of 0.0002 with a single layer having 46 dimension for both the actor and the critic along with RELU activation function, the results obtained are as shown in table 4.6.

Table 4.6 NDCG Score for Advantage Actor Critic with Monte Carlo Update (A2CRank (MC))

K	MQ2007	MQ2008
1	0.4042	0.3715
3	0.3953	0.4100
5	0.4016	0.4625
10	0.4272	0.2457

4.2.6 Proximal Policy Optimization with Monte Carlo Advantage Estimate

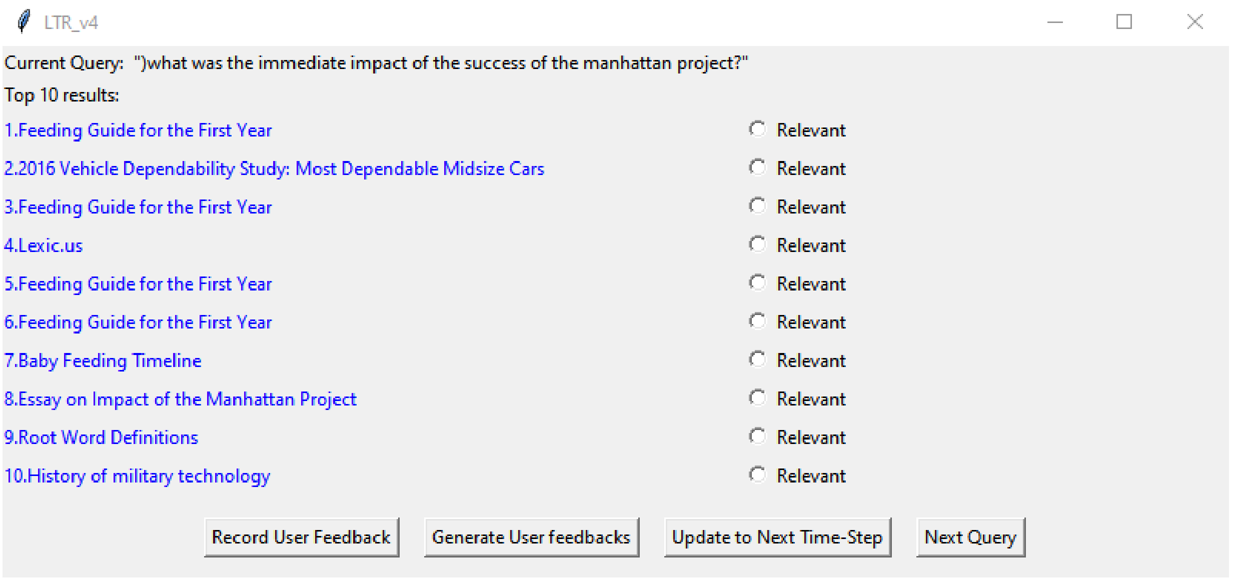
One of the most prominent algorithms used in Reinforcement Learning is the Proximal Policy Optimization algorithm. This when applied to our setting as described in algorithm 3.6 provides results given in table 4.7. The algorithm was tested for 200 iterations using 0.0003 as the actor's learning rate and 0.001 critic learning rate. K epochs were set to 3 and the epsilon clipping to 0.2. The gamma was fixed to 0.99 with the typical value and entropy coefficient of 0.5 and 0.01 respectively. The amount of timesteps to elapse before updating was set to 200. This implementation makes use of 2 neural networks one actor and another critic with a single layer having 64 nodes. TANH was used as the activation function for both of them.

Table 4.7 NDCG Score of PPO with MCAE (PPORank (MCAE))

K	MQ2007	MQ2008
1	0.3779	0.3785
3	0.3950	0.4080
5	0.3951	0.4537
10	0.4218	0.2344

4.2.7 Dynamic Ranking System

The dynamic ranking system is hence constructed by taking a semantic model for the retrieval of documents from the entire corpus and a reinforcement learning agent for the ranking of the documents based on the user behaviour. We observe the interface as shown to the user in Figure 4.1.

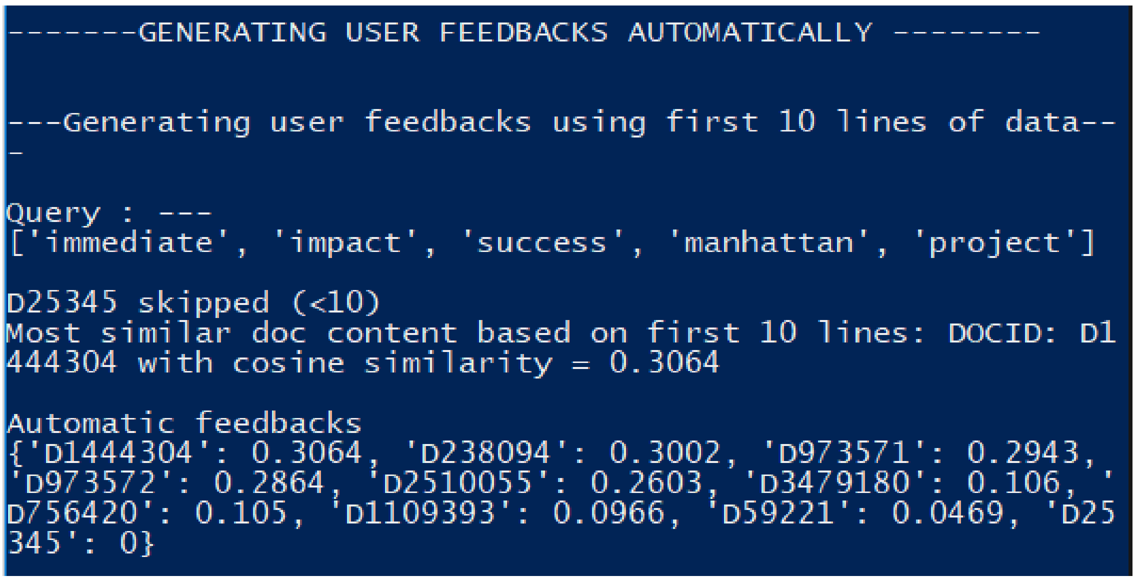


Current Query: ")what was the immediate impact of the success of the manhattan project?"
Top 10 results:
1.Feeding Guide for the First Year ○ Relevant
2.2016 Vehicle Dependability Study: Most Dependable Midsize Cars ○ Relevant
3.Feeding Guide for the First Year ○ Relevant
4.Lexic.us ○ Relevant
5.Feeding Guide for the First Year ○ Relevant
6.Feeding Guide for the First Year ○ Relevant
7.Baby Feeding Timeline ○ Relevant
8.Essay on Impact of the Manhattan Project ○ Relevant
9.Root Word Definitions ○ Relevant
10.History of military technology ○ Relevant

Figure 4.1 IR Ranking simulator

The semantic model used is Doc2Vec model trained on 1.1 million documents. The model is then saved and is noted to be of size 135 Mb. It takes around 15-20 seconds to load the model and for every 10 documents whose cosine similarity are found with respect to the query, the model takes around 2-4 seconds. The Doc2Vec model constructs the paragraph or document embeddings based on a vector size of 50 with minimum number of words as 3, trained for 10 epochs.

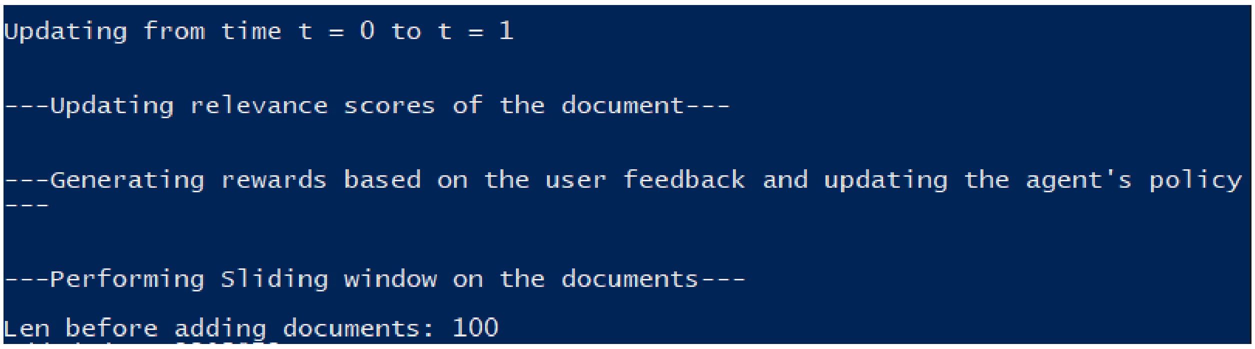
Upon clicking on record user feedback, post click on one of the radio-buttons, the user feedback is recorded manually and given a score of 1.0 indicating its score. Note that the user feedbacks are saved with respect to the query in order to allow for processing of all user feedbacks at once. Similarly, user feedbacks are generated and stored when lacking manual user feedback. This is done by using Doc2Vec to calculate the similarity score on the basis of the first 10 lines of the document shown with respect to the query. This can be seen in the command line as shown in Figure 4.2**.**



```
-------GENERATING USER FEEDBACKS AUTOMATICALLY --------


---Generating user feedbacks using first 10 lines of data--
-

Query : ---
['immediate', 'impact', 'success', 'manhattan', 'project']

D25345 skipped (<10)
Most similar doc content based on first 10 lines: DOCID: D1
444304 with cosine similarity = 0.3064

Automatic feedbacks
{'D1444304': 0.3064, 'D238094': 0.3002, 'D973571': 0.2943,
'D973572': 0.2864, 'D2510055': 0.2603, 'D3479180': 0.106, '
D756420': 0.105, 'D1109393': 0.0966, 'D59221': 0.0469, 'D25
345': 0}
```

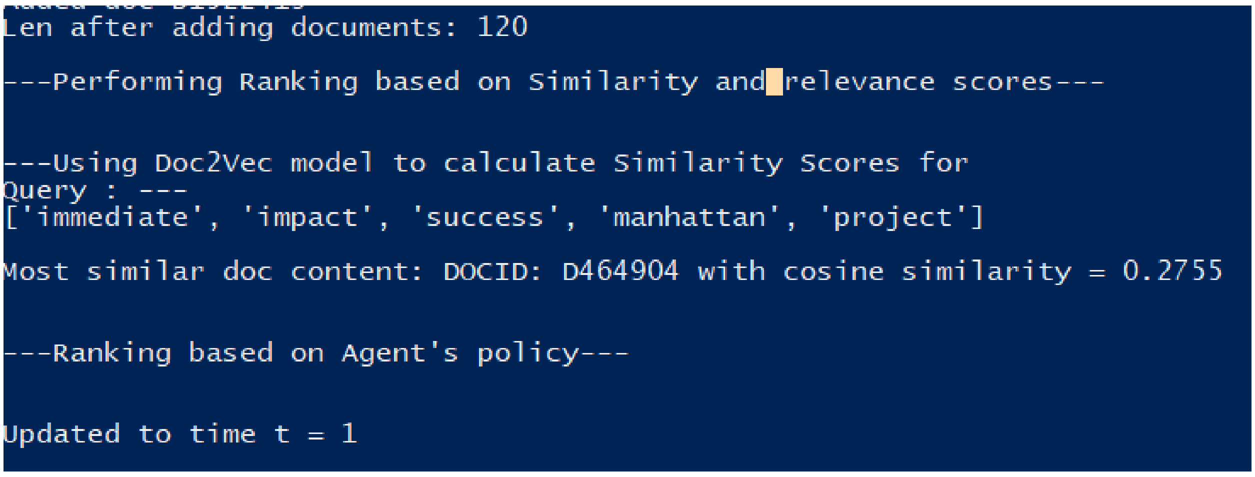
Figure 4.2. Command Line output for Generation of user feedbacks

Thereon, clicking on the “Update to next Time-Step” button, the calling of the agent and the ranking is hence done. During this phase, the relevance scores of the documents are updated by both giving significance to the documents clicked, and decaying the documents not clicked. Post ranking, documents are removed based on a sliding window and documents are added to the set being monitored currently. This is as shown in Figure 4.3 and Figure 4.4



```
Updating from time t = 0 to t = 1

---Updating relevance scores of the document---

---Generating rewards based on the user feedback and updating the agent's policy
---

---Performing Sliding window on the documents---

Len before adding documents: 100
```

Figure 4.3. Updating the dynamic system from time t0 to time t1



```
Len after adding documents: 120

---Performing Ranking based on Similarity and relevance scores---

---Using Doc2Vec model to calculate Similarity Scores for
Query : ---
['immediate', 'impact', 'success', 'manhattan', 'project']

Most similar doc content: DOCID: D464904 with cosine similarity = 0.2755

---Ranking based on Agent's policy---

Updated to time t = 1
```

Figure 4.4. On completion of Updation and after adding N documents

4.3 Significance of the result obtained

The results obtained help us understand how well the reinforcement learning agent performs in ranking tasks or real world problems. This is more apparent when comparing the results with other machine learning models. We can consistently observe that the neural network implementations of the algorithms gave consistent higher scores than their linear versions. In case of MQ2008, the NDCG Score obtained at position 1 and 10 beat the state of the art results while the remaining scores are close to the best scores. This indicates that the actor critic algorithm is an effective algorithm in this problem. With the right hyperparameters and 5 Fold cross validation even higher scores could be achieved. With the effectiveness of the algorithm tested and compared, fitting the algorithm to the model discussed for dynamic ranking would also provide better results. This is shown in table 4.8 and table 4.9

Table 4.8 Comparison of the NDCG Scores achieved for MQ2007 dataset

Algorithm	NDCG@1	NDCG@3	NDCG@5	NDCG@10
RankSVM	0.4045	0.4019	0.4072	0.4383
ListNet	0.4002	0.4091	0.4170	**0.4440**
AdaRank-NDCG	0.3876	0.4044	0.4102	0.4369
SVMMAP	0.3853	0.3899	0.3983	0.4187
MDPRank	0.3954	0.3983	0.4069	0.4350
PPGRank	0.3947	0.3993	0.4089	0.4382
QACRank	0.3694	0.3759	0.3827	0.4120
DQNRank	0.4022	0.4041	0.4069	0.4254
IncrRank	0.3786	0.3871	0.3953	0.4189
A2CRank (MC)	0.4042	0.3953	0.4016	0.4272
PPORank (MCAE)	0.3779	0.3950	0.3951	0.4218
MDPRank Paper Results	**0.4061**	**0.4101**	**0.4171**	0.4416

Table 4.9 Comparison of the NDCG Scores achieved for MQ2008 dataset

Algorithm	NDCG@1	NDCG@3	NDCG@5	NDCG@10
RankSVM	0.3626	0.4286	0.4695	0.2279
ListNet	0.3754	0.4324	0.4747	0.2303
AdaRank	0.3826	0.4420	0.4821	0.2307
RankNet	0.3202	0.3984	0.4408	0.2094
MDPRank	0.3741	0.4099	0.4569	0.2230
PPGRank	0.3741	0.4226	0.4627	0.2290
QACRank	0.3658	0.3920	0.4292	0.2053
DQNRank	0.3376	0.3887	0.4313	0.2320
IncrRank	**0.3969**	0.4148	0.4744	**0.2533**
A2CRank (MC)	0.3715	0.4100	0.4625	0.2457
PPORank (MCAE)	0.3785	0.4080	0.4537	0.2344
MDPRank Paper Results	0.3827	0.4420	0.4881	0.2327
PPGRank Paper Results	0.3877	**0.4511**	**0.4910**	0.2455

Another significance is that using neural network architecture with reinforcement learning gave better results than RankNet which is the non-linear function approximator baseline for the datasets. We also observe that A2CRank, which is advantage actor critic with updates at the end of the trajectory provide better results than the other implemented algorithms for

MQ2007 dataset, and advantage actor critic with TD(0) updates provide better results than the other implemented algorithms for MQ2008 dataset.

4.4 Conclusion

The research was essential in providing an in-depth knowledge about the working of reinforcement learning agents as a ranking model. With the use of these ranking models paired with an accurate and efficient semantic model, one can hope to achieve a robust search engine for query and document retrieval and ranking. The dynamic ranking system shown, shows one such possible model. With the results we could infer that deep reinforcement learning models, that is, reinforcement learning models implemented using a non-linear function approximator are robust, accurate and efficient compared to their linear counterparts. In this section we compared the result with the help of NDCG Scores and to show the validity of the algorithm, performed comparison with the baselines for 2 datasets both belonging to Microsoft's LETOR 4.0 class of datasets[32]. The model used for dynamic learning to rank makes use of the MSMARCO dataset [30], in order to demonstrate the end to end outlook of the possible system. We also noticed that the results achieved with the algorithms are comparable to those obtained by pure machine learning and deep learning techniques and in some cases even outperform.

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

5.1 Brief summary of the work

We have completed an end to end design and implementation of a dynamic search information retrieval system using reinforcement learning that performs LTR. Along with this, we have tested the RL component as a standalone system to experiment and observe the implementations of the paper and further extend to our own implementations. We finally compare 7 models that were implemented to the baselines of the dataset provided. Reinforcement learning hence shows a great potential in this field as most results are improvement over the baselines achieved by other non-reinforcement learning techniques.

5.2 Conclusions

To conclude, as an end result, our comparison scores on the RL models with such vast backgrounds can serve as a wonderful base theory for extension into work that try to perform ranking based on relevance labels. The simulator can be easily used to modify the RL agent and be extended for any future use case. Each algorithm provided an extremely strong foundation to the understanding of the theoretical and practical concepts of Reinforcement Learning and may also serve as a base for beginners willing to understand RL to the core.

5.3 Future Scope of work

We list out possible extensions of our final work here.

1. The GUI interface can be made better with the help of UI / UX concepts or may also be implemented in Java which is more design friendly than GUI implementations in python.
2. The RL models implemented can further be tried with different hyperparameters and variation in the neural network architecture which might enable better scores than the one discussed in the report.
3. The RL models can be tested on larger datasets like the MSLR datasets [31] to observe the learning for a more complex ranking function since the size of the dataset is huge which is beneficial for a neural network.

REFERENCES

Journal / Conference Papers

[1] Crammer, Koby, and Yoram Singer. "Pranking with ranking." In *Nips*, vol. 1, pp. 641-647. 2001.

[2] Nallapati, Ramesh. "Discriminative models for information retrieval." In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 64-71. 2004.

[3] Burges, Chris, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. "Learning to rank using gradient descent." In *Proceedings of the 22nd international conference on Machine learning*, pp. 89-96. 2005.

[4] Cao, Yunbo, Jun Xu, Tie-Yan Liu, Hang Li, Yalou Huang, and Hsiao-Wuen Hon. "Adapting ranking SVM to document retrieval." In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 186-193. 2006.

[5] Joachims, Thorsten. "Optimizing search engines using clickthrough data." In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 133-142. 2002.

[6] Burges, Christopher JC. "From ranknet to lambdarank to lambdamart: An overview." *Learning* 11.23-581 (2010): 81.

[7] Cao, Zhe, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. "Learning to rank: from pairwise approach to listwise approach." In *Proceedings of the 24th international conference on Machine learning*, pp. 129-136. 2007.

[8] Xu, Jun, and Hang Li. "Adarank: a boosting algorithm for information retrieval." In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 391-398. 2007.

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

[10] Zhou, Jianghong, and Eugene Agichtein. "RLIRank: Learning to Rank with Reinforcement Learning for Dynamic Search." In *Proceedings of The Web Conference 2020*, pp. 2842-2848. 2020.

[11] Hu, Yujing, Qing Da, Anxiang Zeng, Yang Yu, and Yinghui Xu. "Reinforcement learning to rank in e-commerce search engine: Formalization, analysis, and application." In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 368-377. 2018.

[12] Silver, David, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. "Deterministic policy gradient algorithms." In *International conference on machine learning*, pp. 387-395. PMLR, 2014.

[13] Wei, Zeng, Jun Xu, Yanyan Lan, Jiafeng Guo, and Xueqi Cheng. "Reinforcement learning to rank with Markov decision process." In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 945-948. 2017.

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

[15] Konda, Vijay R., and John N. Tsitsiklis. "Actor-critic algorithms." In *Advances in neural information processing systems*, pp. 1008-1014. 2000.

[16] Mnih, Volodymyr, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. "Asynchronous methods for deep reinforcement learning." In International conference on machine learning, pp. 1928-1937. PMLR, 2016.

[17] Xu, Jun, Zeng Wei, Long Xia, Yanyan Lan, Dawei Yin, Xueqi Cheng, and Ji-Rong Wen. "Reinforcement Learning to Rank with Pairwise Policy Gradient." In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 509-518. 2020.

[18] Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal policy optimization algorithms." *arXiv preprint arXiv:1707.06347* (2017).

[19] Schulman, John, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. "Trust region policy optimization." In *International conference on machine learning*, pp. 1889-1897. PMLR, 2015.

[20] Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient estimation of word representations in vector space." *arXiv preprint arXiv:1301.3781* (2013).

[21] Le, Quoc, and Tomas Mikolov. "Distributed representations of sentences and documents." In *International conference on machine learning*, pp. 1188-1196. PMLR, 2014.

Books

[22] Manning, Christopher D.; Raghavan, Prabhakar; Schütze, Hinrich. *"*Introduction to Information Retrieval*"*. Cambridge University Press.2018.

[23] S. Grace and P. Gravestock, "Inclusion and diversity: Meeting the needs of all students. Routledge", 2008.

[24] R. A. of Engineering, "Engineering ethics in practice: a guide for engineers". Royal Academy of Engineering London (UK), 2011.

[25] I. Van de Poel and L. Royakkers, "Ethics, technology, and engineering: An introduction". John Wiley & Sons, 2011.

[26] M. R. Spiegel, J. J. Schiller, and R. A. Srinivasan, *Schaum's outlines probability and statistics*. McGraw-Hill, 2013.

Web

[27] NeurIPS | 2021.[Online].Available: https://nips.cc/

[28] SIGIR 2021.[Online].Available: https://sigir.org/sigir2021/

[29] ICML | 2021 |Thirty-eighth International Conference on Machine Learning. [Online]. Available: https://icml.cc/

[30] MSMARCO dataset. [Online]. Available: https://microsoft.github.io/msmarco/

[31] Microsoft Learning to Rank Datasets. [Online]. Available:
https://www.microsoft.com/en-us/research/project/mslr/

[32] LETOR: Learning to Rank for Information Retrieval. [Online]. Available:
https://www.microsoft.com/en-us/research/project/letor-learning-rank-information-retrieval/

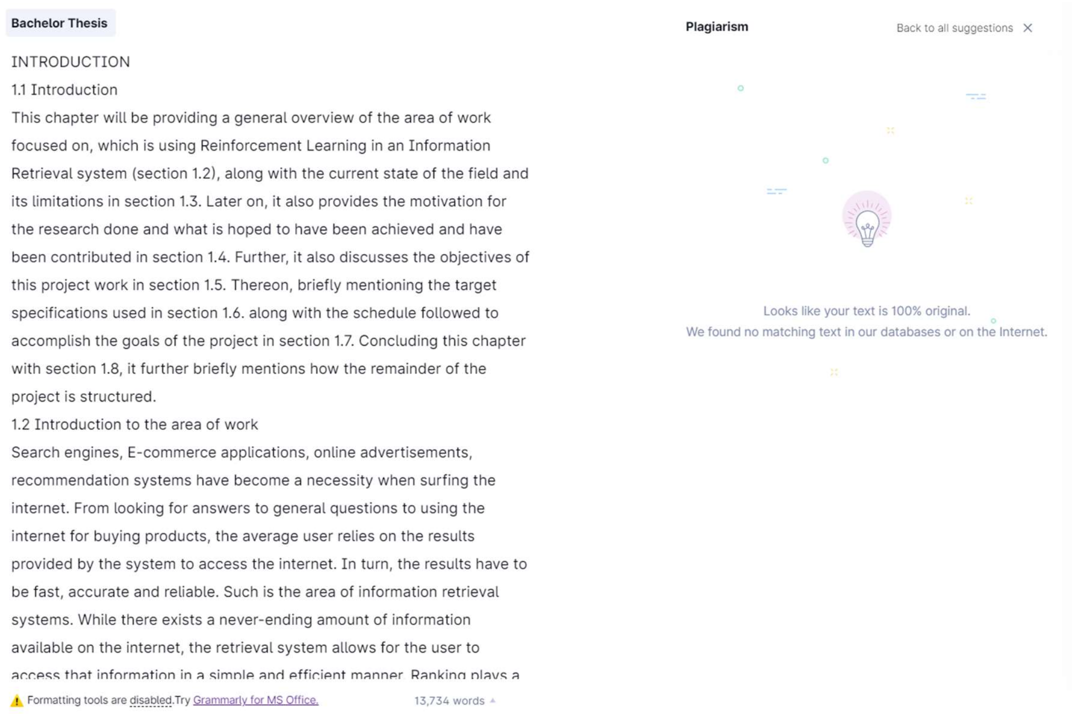
[34] The ABCs of Learning to Rank.[Online].Available:
https://lucidworks.com/post/abcs-learning-to-rank/

[35] Deep Reinforcement Learning - Julien Vitay..[Online].Available:
https://julien-vitay.net/deeprl/Valuebased.html#sec:deep-q-network-dqn

[36] Part 2: Kinds of RL Algorithms — Spinning Up documentation. (2021). [Online]. Available: https://spinningup.openai.com/

[37] TREC dataset.[Online].Available: https://trec.nist.gov/data.html

Plagiarism Report



Bachelor Thesis

INTRODUCTION
1.1 Introduction
This chapter will be providing a general overview of the area of work focused on, which is using Reinforcement Learning in an Information Retrieval system (section 1.2), along with the current state of the field and its limitations in section 1.3. Later on, it also provides the motivation for the research done and what is hoped to have been achieved and have been contributed in section 1.4. Further, it also discusses the objectives of this project work in section 1.5. Thereon, briefly mentioning the target specifications used in section 1.6. along with the schedule followed to accomplish the goals of the project in section 1.7. Concluding this chapter with section 1.8, it further briefly mentions how the remainder of the project is structured.
1.2 Introduction to the area of work
Search engines, E-commerce applications, online advertisements, recommendation systems have become a necessity when surfing the internet. From looking for answers to general questions to using the internet for buying products, the average user relies on the results provided by the system to access the internet. In turn, the results have to be fast, accurate and reliable. Such is the area of information retrieval systems. While there exists a never-ending amount of information available on the internet, the retrieval system allows for the user to access that information in a simple and efficient manner. Ranking plays a

Formatting tools are disabled.Try Grammarly for MS Office.   13,734 words

Plagiarism   Back to all suggestions

Looks like your text is 100% original.
We found no matching text in our databases or on the Internet.

PROJECT DETAILS

Student Details			
Student Name	**Roshan Jacob Manoj**		
Register Number	170905450	Section / Roll No	C / 58
Email Address	roshan.mjacob99@gmail.com	Phone No (M)	8369505016
Project Details			
Project Title	**Deep Reinforcement Learning Based Information Retrieval**		
Project Duration	6 months	Date of reporting	4/01/2021
Organization Details			
Organization Name	**Defense Research and Development Organization**		
Full postal address with pin code	DRDO Complex, C V Raman Nagar, Bengaluru, Karnataka 560093		
Website address	https://www.drdo.gov.in/labs-and-establishments/centre-artificial-intelligence-robotics-cair		
External Guide Details			
Name of the Guide	**Mr. Ashish Srivastava**		
Designation	Scientist Centre Artificial intelligence Robotics DRDO		
Full contact address with pin code	DRDO Complex, C V Raman Nagar, Bengaluru, Karnataka 560093		
Email address	ashish.srivastava2105@gmail.com	Phone No (M)	7760847589
Internal Guide Details			
Faculty Name	**Dr. Siddhalinga Swamy P.C.**		
Full contact address with pin code	Dept of Computer Science & Engg, Manipal Institute of Technology, Manipal – 576 104 (Karnataka State), INDIA		
Email address	pcs.swamy@manipal.edu		