Chapter 4

A Multi Agent Deep Reinforcement Learning based approach for large scale Learning to Rank

Reinforcement Learning based appraoches have been used effectively for the Learning to Rank problem; however, work on a Multi-Agent-based framework for the same has been rather limited. The present Learning to Rank techniques over looks the correlation between the documents by overlooking shared information between them. The Multi-Agent framework is used to capture the correlation between documents by sharing information among multiple agents, with the centralised critic framework having access to this global information. Furthermore, traditional Reinforcement Learning approaches deal with high variance and noise. These effects are exacerbated in huge datasets with millions of documents. We use the Deep Reinforcement Learning actor-critic architecture to address these challenges, which learns value estimations directly from samples. Deep Policy Gradient-based techniques have proven to be more efficient in large action space situations, i.e., with millions of items, by leveraging neural networks' powerful approximation capabilities. The query document pair information is used by most existing Learning to Rank algorithms, but the shared information between the documents is ignored.

We intend to use an interactive model with Multi-Agent Reinforcement Learning to extract and utilize the correlation between documents. The agents share their observations about the environment in an attempt to learn a coordinated model. Sharing the global state across agents increases correlation, which is important for learning a coordinated strategy [129]. Consequently, by sharing information about the documents across the different agents, we presume that different agents would learn the link between the documents and the coordination of their policies. The environment provides the agents with the global state, rewards, and cooperative actions. The association between distinct agents can be further examined depending on the variance in reward of different agents [129]. We base our reward structure, which provides a higher payoff for the more relevant docu- ments, on the notion that training several agents in conjunction with one another over time will yield similar documents with higher rewards that are more relevant to the query.

A shared policy helps reduce computational issues in a multi-agent context with centralised critic algorithms, which suffer from the curse of dimensionality as the number of agents increases. The environment does not go to the next state until all of the agents have completed their current policy actions.

Reinforcement Learning based algorithms have been successfully applied to the Learning to Rank problem in recent years [41, 110, 85]. Although policy gradient approaches have been widely used for the Learning to Rank [121, 68, 32, 139], the majority of existing policy-based methods also deal with high variance and gradient noise. Due to sampling from distinct trajectories during learning, monte-carlo updates create substantial variance in policy gradient approaches [92]. Multi Armed Bandit-based systems for Learning to Rank using clicks as implicit feedback have also been presented [77, 123, 89], albeit clicks are thought to be noisy signals due to pervasive position bias [71]. A Reinforcement Learning based approach using query formulation was proposed in [92]; however, it used semantic matching between the terms and query, that fails to capture different relevance signals [25]. A multi agent based approach was proposed in [139]. However, it treats all documents as the

agents in Multi-agent settings, which can quickly become intractable in large scale real scenarios. Traditional RL algorithms, on the other hand, are not scalable to large state-action spaces due to the curse of dimensionality [21].

Finding relevant documents for the query has become an essential task in information retrieval, owing to the vast web search space. Policy gradient approaches have been shown to be effective in problems with large action spaces, i.e., millions of items [16], as they utilize parameterized policies tolearn optimal policy from sampled data directly. Value-based techniques, on the other hand, involve calculating value function estimates for every actions. We used the Deep Reinforcement Learning method to address such large scale ranking problems as it has been proven to be highly efficient in complex scenarios that were intractable with regular Reinforcement Learning approaches [67]. With its powerful function approximation capabilities, the deep neural network structure has been successful in problems with huge state-action spaces [21].

RL based approaches generally experience curse of dimensionality issue in a single agent setting with large scale models [116, 128]. One approach is to use parallel computation by deploying numerous agents with comparable tasks to increase learning efficiency and stability [57]. Although deep neural network-based models have enhanced the scalability of reinforcement learning based methods, the singular centralized RL framework still suffers from high dimensionality in large-scale scenarios [23]. The actor-critic technique reduces bias and variation in policy-based algorithms by parameterizing the model to construct the value function [48]. To address challenges such as as scalability and partial observability in multi-agent environments, the Centralized Training with Decentralized Execution (CTDE) paradigm has been proposed [135]. It operates on two main principles: centralized training and decentralized execution. During the training phase, all agents share information and have access to a global view of the environment, which enhances the efficiency of their learning process. Once training is complete, agents operate independently based on their local observations, allowing them to adapt to dynamic environments

without relying on centralized control. This approach offers significant benefits, including improved learning through collective experiences and robustness, as agents can respond to changes or failures without oversight. CTDE/MADDPG appraoch has demonstrated that offering more information to critics during training leads to agents developing coordinated actions/behaviors [64]. Furthermore, centralized training has been shown to be effective by lowering variance [27], being more robust [53, 15], or giving better stability during training [88]. The centralized planning and execution framework is a fundamental impediment for cooperation between the agents in large-scale multi-agent situations. We solve this issue by training a centralized stochastic RL policy while executing it in decentralized way; hence, during the execution stage, local observations and actions are not communicated across the agents. MARL solves the scalability problem by distributing global information and control to each local agent [24]. In general, multi-agent systems frameworks are created to make easy to add more agents, making them inherently scalable. Gathering enough data is one of the primary barriers for RL algorithms to learn an effective policy. To address this issue, we use the experience replay technique, which saves the experience tuples in memory and then randomly samples from them repeatedly. With several uses of the experience tuple from the buffer randomly, experience replay leads to more reliable training of a deep neural network and also improves sampling efficiency. the Multi-Agent framework involves multiple agents (or models) that work collaboratively to improve the ranking of documents. Each agent can be seen as an individual component that contributes to the overall ranking process.

The shared information mechanism can be utilized to capture the correlation between the documents in the multi agent framework. Each agent in the framework can be specialized or focused on different aspects of the ranking task. For example, one agent might focus on document features, while another might focus on user behavior data. These agents share information with each other, which allows them to leverage diverse perspectives and insights. For instance, an agent that excels at understanding user preferences can share its findings with an agent focused on

content relevance. Further, the Centralized Critic has access to the global information collected by all agents. It acts as an overseer, evaluating the performance of the individual agents and providing feedback to guide their learning processes. The centralized critic benefits from the aggregated information and can offer a more holistic view of the ranking task, which helps in improving the overall performance of the ranking system.

The advantages of shared information mechanism in the multi agent framework can be listed as:

Diverse Insights: Multiple agents can provide varied insights and perspectives, improving the robustness of the ranking system. Specialization: Different agents can specialize in different features or aspects of the documents and user interactions, potentially leading to more nuanced and effective ranking.
Collaborative Learning: By sharing information, agents can learn from each others strengths and weaknesses, leading to better overall performance.
Improved Feedback: The centralized critic, with access to global information, can provide more informed and comprehensive feedback, which enhances the learning process for each agent.
Coordination: Effective coordination between agents and the centralized critic is crucial to ensure smooth information sharing and learning. Scalability: Managing and integrating information from multiple agents can become complex, so scalable solutions are necessary.
Efficiency: The system should be designed to ensure that the overhead of managing multiple agents and a centralized critic does not outweigh the benefits.
Thus, multi-agent and centralized critic approach provides a sophisticated way to handle complex ranking tasks by leveraging collaborative learning and comprehensive feedback mechanisms.

To address these issues, we propose a Multi-Agent RL algorithm for Learning to Rank based on the MADDPG approach. The multi-agent design offers significant

benefits such as coordinated activities and stability via a centralized critic [64]. MARL enables agents to design optimal coordination rules among themselves without developing a full decision model of the environment [126]. Instead, it allows diverse agents to learn and explore the underlying environment by changing their actions to the dynamic environment. In contrast to using a single-agent framework, learning of numerous agents in a shared environment has proven to be more beneficial due to the extra knowledge gained from estimating the policies of other agents [36, 119]. Furthermore, during the learning process regarding real-world scenarios, RL based approaches incorrectly assume circumstances such as full observability and full network accessibility. MADDPG further stabilizes the learning process by overcoming the non-stationarity issue and lowering the variance in Q-value estimates. For modelling our problem, we utilize the CTDE framework. Using CTDE framework, agents can share the parameters of a single value network and a replay buffer, thereby reducing the algorithm's policy search space because it is now optimising a single policy rather than many policies for each agent. This strategy also addresses the non-stationarity issue in multi-agent training by exposing the policy to all of the agents' experiences at the same time. The primary idea behind CTDE algorithms is to supply more extensive state information to agents during training phase so that their individual experiences can be aggregated through centralization of training, allowing agents to learn more effectively [86]. This allows agents to generate coordinated actions after learning is complete, on the basis of incomplete local observations, i.e., partially observed states [99]. Furthermore, to address the non-stationarity issue, each agent can learn an approximation of the online policies of other agents. As a result, we're learning multiple policies, one for each agent.

MADDPG is a multi-agent RL approach that expands on the single agent actor-critic solution by providing the critic with information about the policies of other agents. To address the scalability issue, we use the CTDE framework. MADDPG employs several critics to learn the joint policy for multiple agents, with each critic having access to the other agents' observations, actions, and target policies throughout

training. The role of each critic is to integrate all states-actions to be used as input and obtain the relevant Q-value estimations for each agent's actor network. The critic receives the actions of several agents as well as global state information and returns the Q-value estimate to each actor. Each agent in the MADDPG system has a centralised critic and a decentralised actor.

The MADDPG algorithm learns the value function through centralised training. As a result, the critic has access to all other agents' behaviours, allowing it to evaluate the influence of the shared policy on each agent's long-term reward. MADDPG framework consists of n agents, each with an actor and critic network. Figure 4.1 depicts the architecture. During training, the critic module receives global input from different agents' actors and critic parameters to learn the model, however during execution, the centralised module is shut off and each agent has access to only its own module.

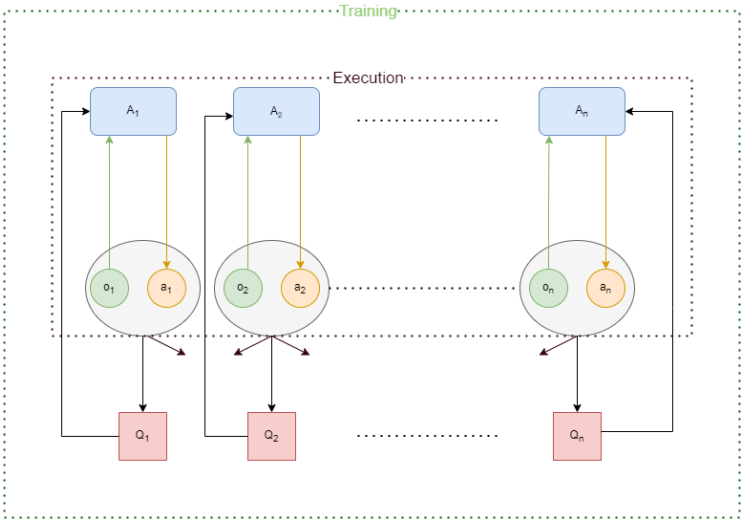

Figure 4.1: MADDPG Architecture

4.1 Problem Statement

The Learning to Rank problem consists of up of several queries, each of which is associated with a separate document and its ground truth labels. We again describe the problem as $\{V_i, X_i, Y_i\}_{i=1,2,\ldots}$, where Q_i denotes the query and $X_i = \{X_1, \ldots, X_k\}$ and $Y_i = \{Y_1, \ldots, Y_k\}$, represents the candidate documents and the

corresponding relevance labels associated with them, respectively. Once the user submits a query, they are provided with the relevant list of documents. The objective is to rank the candidate documents according to their relevancy and provide the output list to the user.

4.2 Multi Agent MDP Formulation

We used a multi-agent RL scenario to represent the Learning to Rank task, with each agent being an independent MDP module with its own local observations, actions, and rewards. The CTDE framework is used to model the Multi Agent RL configuration for the ranking job. The fundamental reason for the centralised critic to use joint state action pairs for parameter updating is that even if the agent's policies change, the environment remains stationary if we have knowledge of all of the agents' actions. Despite the fact that the agents only have access to local observations and actions, the usage of centralised critics during training for all agents offers information on the optimality of their actions for updating their policies. This reduces the impacts of non-stationarity while preserving policy learning at a lower state space. During the update, we utilise each agent's target network to compute the next action to improve training stability. We change the target's actor parameters periodically to match the agent's actor settings. The algorithm also abandons the assumption that each agent is automatically aware of the actions of other agents; instead, it infers other agents' policies to improve autonomy and exploration.

In our work, we model the Learning to Rank problem using the Markov Decision Process framework. In a shared environment, we have n different agents interacting with each other. Each agent performs a specific action in accordance with its own policy and then obtains reward from the environment. We describe the Multi Agent MDP setting as:

Observation : Each agent i in our method has its own observation space o_i of the

environment. All of the agents share the same environment but only have access to their own observations. The local observation state o_t at time t is represented as $[X_t, Y_t]$, where X_t is the list of k documents and Y_t is the candidate set of all items.

Each document is represented by a 136-valued feature vector(Section 4.4). To create the observation space, we first feed a vector list of $k-1$ documents into a matrix, which are the latest actions that other agents choose for rating. After obtaining the list of documents, the matrix is sent through a Convolutional Neural Network layer [114] in order to extract the sequential patterns between the documents by capturing picture features [103] . The CNN module's output is next passed through two fully-connected ReLU network layers to get the higher-level features, v_t. The current observation state o_t^i is then obtained by doing the cross product with the preceding observation vector o_{t-1}^i, as shown in Figure 4.2. To begin, the prior observation vector o_0^i is treated as a random vector between 0 and 1. The joint observation space is denoted by $O_t = \{o_t^1, \ldots, o_t^n\} \in \mathcal{O}$. In our problem formulation, we define the system state as the sum of all the agents' observations o_i.

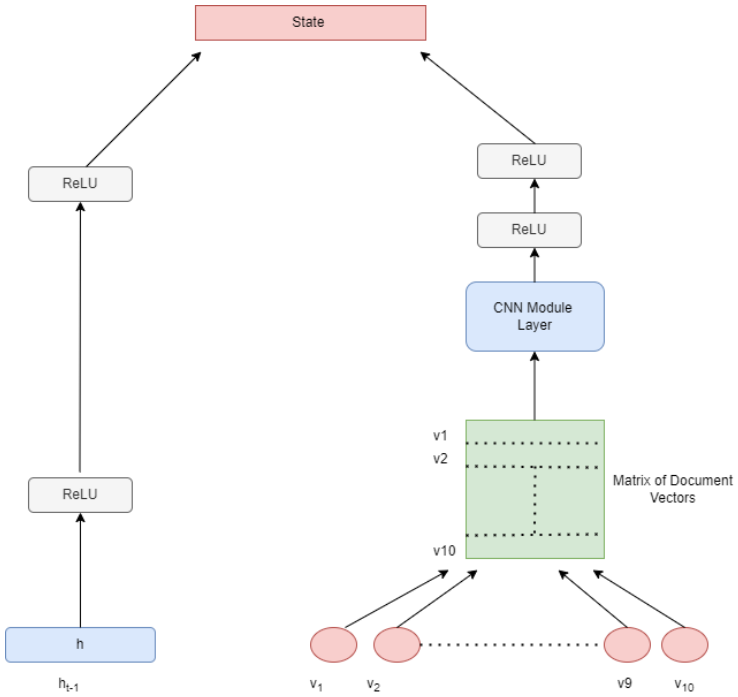

Figure 4.2: State/Observation representaion module

Action : For every agent i, we generate the action from the observation o_t^i in two

stages as deterministic policy gradient operates with continuous action space. A sub-action is represented by a continuous vector a_i* generated by the actor-network. We construct the action by mapping the sub-action to the real action in action space, a_t^i from this, and selecting the one with the highest value by computing the product of the generated vector with all the document vectors.

Reward : The agent then receives feedback from the environment after it generates the action. The reward can be understood as an evaluation of the selected document's relevance to the query. The reward for agent i is estimated as the increase in the value of DCG at time step t and is calculated as : $\mathrm{r_{i_{(o_t,a_t)}}} = \frac{2^{rel_{a_t}}-1}{\log_2(t+1)}$, where rel_{a_t} denotes the relevance of the selected document. The reward function can directly optimize the IR evaluation metric using this approach, which has been proven to be effective in recent works.

Policy : Each agent has it's own actor network that implements the policy. The agent sees only to it's own observations with it's policy implementing the mapping from local observation to the actions. The policy for the agent i can be described as $\mu_i : o_t^i \rightarrow a_t^i$.

Discount factor : The discount factor $\gamma \in [0, 1]$ measures the trade-off between current and future rewards. When γ is set to zero, the agent considers only the present reward and ignores the future rewards. Similarly, when γ is set to 1, the agent considers all future as well as immediate rewards.

4.2.1 Actor Critic Module

In our scenario, each agent has its own actor and critic network. Furthermore, each agent has an observation space and a continuous action space that are not shared with other agents. Using the Markov Decision Process paradigm (Section 1.1), we describe the individual observation and action space for each agent. Actors are deterministic policy learning agents who solely use local observations. This mitigates the impact of a rapidly expanding state and action space. For greater

stability during training, the target network, which is a time delayed duplicate of the original network for the actors, is used. The critic network then estimates Q-values based on the joint state and action values. During training, the critic continues to learn the combined action-value function.

Let π_θ represent a policy with parameters θ, and $J(\pi_\theta)$ denote the expected finite-horizon undiscounted return of the policy. The objective of the Policy Gradient method is to find the optimal θ which maximizes the cumulative return over all trajectories. It can be stated as $J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\mathbb{R}(\tau)]$, where τ is the trajectory and $R(\tau)$ representing the reward over trajectory.
The gradient of $J(\pi_\theta)$, also knows as Naive Policy gradient [109] is given as,

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_\pi[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) Q^\pi(s,a)] \tag{4.1}$$

For multi-agent settings, we use the Policy Gradient equation described above. Let there be N agents in the system with the deterministic policies $\mu = \{\mu_1, \ldots, \mu_N\}$ paramaterized as $\theta = \{\theta_1, \ldots, \theta_N\}$. We can then write the gradient of the expected return for agent i as,

$$\nabla_{\theta_i} J(\mu_i) = \mathbb{E}_{z,a\sim\mathcal{B}}[\nabla_{\theta_i}\mu_i(a_i|s_i) \nabla_{a_i} Q_i^\mu(z, a_1, \ldots, a_N)|_{a_i=\mu_i(o_i)}], \tag{4.2}$$

, with $Q_i^\mu(x, a_1, \ldots, a_N)$, denoting the centralized action-value function and z, representing the system's state that comprises the observations of all the agents, i.e., $z = \{o_t^1, \ldots, o_t^n\}$ at time t. The function receives the system state, z, and all agent actions, $a_1 \ldots, a_n$, as inputs, and returns the Q-value for the agent i.

The experience replay buffer $\mathcal{H}$, that stores experiences of all the agents, is used for efficient off policy learning. It contains tuples $(s, s', a_1, \ldots, a_n, r_1, \ldots, r_n)$, i.e. joint state, joint action, joint next state and the individual rewards corresponding to every agent. The replay buffer assists in randomizing the samples extraction and smoothing over sample data distribution leading to better convergence and stability

during the training [110]. At every iteration, we then sample a batch of tuples from the replay buffer to train our agents.

The action-value function Q_i^{μ} is then updated with one step lookahead TD error as,

$$\mathcal{L}(\theta_i) = \mathbb{E}_{z,a,r,z'}[Q_i^{\mu}(z, a_1, \ldots, a_N) - w^2], w_i = r_i + \gamma Q_i^{\mu'}(z', a'_1, \ldots, a'_N)|_{a'_j = \mu'_j(o_j)}] \tag{4.3}$$

, with $\mathcal{L}$ denoting the loss function and $\mu' = \{\mu_{\theta'_1}, \ldots, \mu_{\theta'_n}\}$ denoting the set of target policies with parameters θ'_i and w_i represents the q-value. The q-value estimates are provided to different agent's actor networks throughout time to support the training process. Additionally, we use the deterministic policy gradient to update the actor parameters of each agent.

$$\nabla_{\theta_i} J(\theta_i) = \sum_j \nabla_{\theta_i} \mu_i(a_i|s_i) \nabla_{a_i} Q_i^{\mu}(z, a_1, \ldots, a_N)|_{a_i = \mu_i(o_i^j)} \tag{4.4}$$

, where μ represents the agent's actor. As a result, we take the gradient with respect to the actor's parameters, led by the Q-values from the centralised critic.

Further, every agent maintains an approximation $\hat{\mu}_{\phi_i^j}$, with ϕ denoting the approximation parameters to the agent's policy μ_j, to relax the assumption that each agent is aware of every other agent's policy during learning phase. The agents use observations from $n-1$ more networks for estimating the correct policy of other agents through policy approximation. The approximation policy is realized by using the entropy regularizer to maximize the log likelihood of each agent's actions [132]. With the approximate policies, w in Eqn. 4.3 can be approximated as $\hat{w}$:

$$\hat{w} = r_i + \gamma Q_i^{\mu'}(z', \hat{\mu}_i'^1(o_1), \ldots, \hat{\mu}_i'^N(o_1)) \tag{4.5}$$

, with $\hat{\mu}_i'^j$ representing target network for the approximate policy $\hat{\mu}_i^j$.

4.3 Algorithm

In this section, we propose the algorithm, MA-DRLRank 3, based on the MADDPG framework, addressing the various issues described ABOVE. We employ the multi agent architecture to capitalize on the document correlation to generate coordinated actions. The algorithm follows the CTDE approach, in which the agents share information during training but only deal with local observations during evaluation. It uses the query document pairs and the related relevance labels as the training set and initializes the actor and critic network parameters for the agents. Furthermore, we make the target network's parameters similar to the main network's (Line 1-2). Furthermore, we keep the target network's parameters similar to the main network's (Line 1-2). Following this, we initialize the noise as well as the initial state from the environment (Line 4-5). The algorithm then chooses an action for each agent i based on its current policy, μ_i (Line 6-8). Following the execution of the action, the agent obtains the reward r_i from the environment and next state s' is generated (Line 9). Further, the joint state for all agents, s', actions , rewards and the next state is stored in the experience replay buffer $\mathcal{H}$ (Line 10). After the episode is over, each agent draws a random experience tuple $\mathcal{S}$ from the replay buffer (Line 13). We then employ the approximation over the policies of other agents to derive the revised Q-value, w, with $\hat{\mu}_i^{'i}(o_i)$, denoting the target network policy (Line 14). The parameters of the actor and critic are then updated. We update the parameters of the critic by minimising the loss between w_i' and Q_i^μ (Line 15). Similarly, we use the sampled gradient ascent approach to update the policy parameters for the actor's network (Line 16). Finally, we change the agents' target network settings (Line 18). In the execution stage, we also apply a softmax layer over the actor and use probability score as the confidence value over the item. Finally. we sort the documents by confidence score to present the final ranking.

Algorithm 3 MA-DRLRank

Input: Query-Document Pairs $\{Q_i, X_i, D_i\}$, initialize policy parameters θ_i, Q-function parameters μ_i, empty replay buffer $\mathcal{H}$
Initialize target networks for every agent θ'_i
for episode in *range* 1 to e do
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial state S
    for t in *range* 1 to episode length do
        Select action $a_i = \mu_{\theta_i}(o_i) + \mathcal{N}$ for every agent i, according to policy π_i
        For each agent i, execute the action a_i, s.t, $a_t = a_t^1, \ldots, a_t^N$
        Observe reward $r_t = r_t^1, \ldots, r_t^n$ and next state , s'
        Store the tuple (s, a, r, s') in the experience replay buffer $\mathcal{H}$,
        $s \leftarrow s'$
        for each agent $i = 1$ to N do
            Sample a random batch of m samples (s^k, a^k, r^k, s'^k) from the replay buffer $\mathcal{H}$
            Set $w^k = r_i^k + \gamma Q_i^{\mu'}(z', a'_1, \ldots, a'_N)|_{a'_j = \hat{\mu}'_j(o_j)}]$
            Update the critic network by minimizing loss $\frac{1}{m}\sum_k((w - Q_i^{\mu}(z^k, a_1^k, \ldots, a_N^k))^2)$
            Update the actor network with the sampled policy gradient,

$$\nabla_{\theta_i} J(\theta_i) = \sum_k \nabla_{\theta_i} \mu_i(a_i|s_i) \nabla_{a_i} Q_i^{\mu}(z, a_1^k, \ldots, a_N^k)|_{a_i = \mu_i(o_i^j)}$$

        end for
        For every agent i, update the target network parameters :

$$\theta'_i \leftarrow \rho\theta_i + (1 - \rho)\theta'_i$$

    end for
end for

4.4 Experiments

We performed the experimentson two large scale Microsoft Letor datasets, MSLR-Web30k and MSLR-Web10k. The two datasets encapsulate the relationship between queries and documents by combining feature vectors generated from query-url document pairs with relevance judgement labels. MSLR-Web30k has 30000 searches and 3771125 document url pairs, whereas MSLR-Web10k contains 10000 queries and 125000 document pairings. Each instance of a query-document pair is represented by 136 numerical attributes. Furthermore, each query document pair is assigned a relevance label ranging from 0(least relevant) to 4(most relevant), indicating the relevance to the query. Table 4.1 has a detailed description of the two datasets. For instance, $relevance\_label, query\_id, feature-1, feature-2, ..., feature-136$ denotes a vector with query id with its corresponding features and a relevance label. The various features indicates document's TF-IDF, query term number, query term ratio, IDF, BM-25, Pagerank, SiteRank, URL dwell time, etc. We compared our experimental results to the following state-of-the-art baselines RankSVM [107], RankNet [125], AdaRank [127], Setrank [73] and MDPRank [54]. We also compared it to our first algorithm, DRLRank. We employed neural network-based structures to implement Ranknet, Listnet, and Adarank. For the neural network, we employed two hidden layers and varied the number of hidden layer nodes from 50 to 200. The Adam optimizer was used to train the multiple baselines using the corresponding ranking loss functions of each model. The hyperparameters for the various baselines are selected using a grid search from the values listed in Table 4.2. For training the RL agent, we chose a learning rate of 10^{-3} and a discount factor of 0.99. For our method, we employed a learning rate of 10^{-3} for the actor and 10^{-4} for the critic, as well as a variable discount factor. In all trials, we utilised standard cross-validation on these datasets and provided the average results. The studies were carried out using an Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz processor, x86_64 architecture, with 264 GB RAM.

Evaluation metrics : We use two general evaluation measures in IR to examine the performance of the proposed algorithm on the datasets., Normalized Discount Cumulative Gain (NDCG) [122] and Mean Average Precision (MAP)[59]. In the experiments, we used NDCG at a position 1, 5 and 10 were used for evaluation.

Table 4.1: Statistics for MSLR Web 10k and MSLR Web 30k Datasets

Dataset	Queries	Documents	Features	RelevanceLabels
MSLR-Web30K	18,919	3771125	136	5
MSLR-Web10k	784	1200000	136	5

Table 4.2: Hyperparameters

Dropout Rate	0	0.2	0.4	0.8
Learning Rate	10^{-2}	10^{-3}	10^{-4}	10^{-5}
Epochs	200	400	800	1600
Batch Size	64	126	256	512

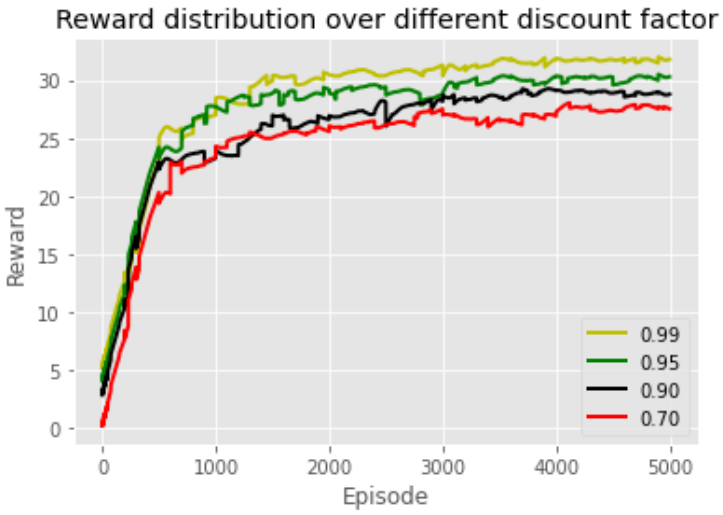

Figure 4.3: Reward Distribution with varying Discount Factor on MSLR Web 30k

4.5 Experimental Results

The experiments were conducted on two large scale datasets, MSLR-Web10k and MSLR-Web30k. The results for the two experiments are shown in Tables 4.3 and Table 4.4, respectively. With the exception of NDCG@3 and NDCG@7, our approach outperforms the different baselines for most ndcg metrics in the MSLR-web-10k

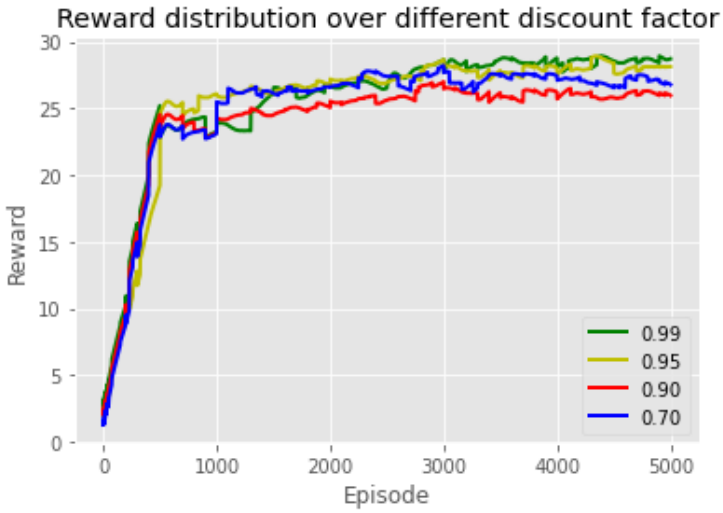

Figure 4.4: Reward Distribution with varying Discount Factor on MSLR Web 10k

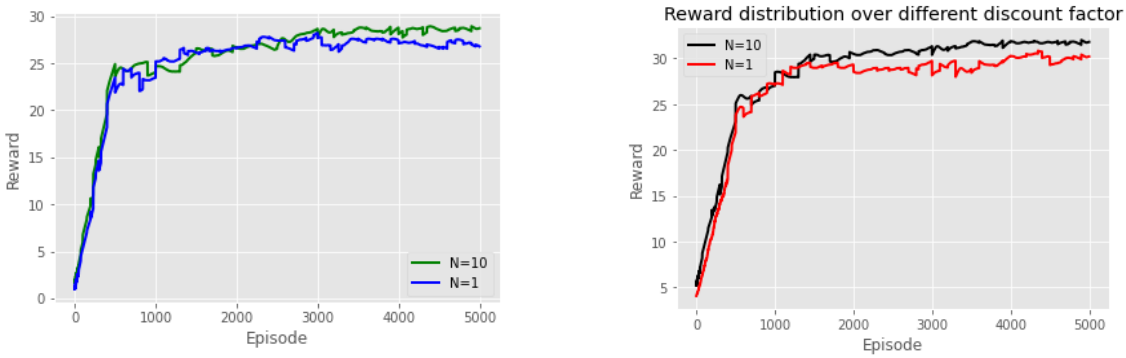

Figure 4.5: Performance of the algorithm compared with the single agent version on MSLR-Wb 10k (left), MSLR-Web 30k (right), N denotes the number of agents

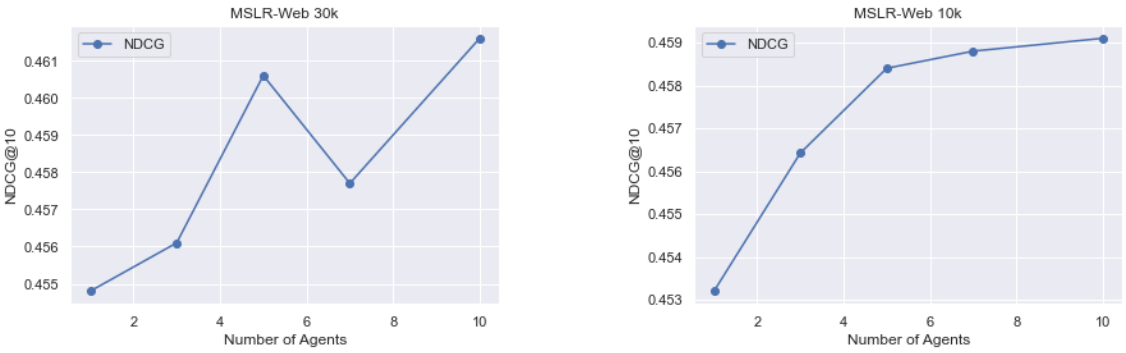

Figure 4.6: NDCG metric for different number of agents (a)MSLR -Web 10k (b) MSLR-Web30k

dataset. Similarly, apart from NDCG@7, our approach outperforms other baselines on the MSLR-web-30k dataset. Our approach performs better on the MSLR Web 30 k dataset than the MSLR Web 10k dataset, based on the observation from results of the two datasets. We attribute this improvement in performance to the fact that

Table 4.3: Results for different metrics on MSLR Web-10k Dataset

Method	NDCG@1	NDCG@3	NDCG@5	NDCG@7	NDCG@10	MAP
RankSVM	0.4262	0.4321	0.4280	0.4387	0.4320	0.4304
Ranknet	0.4321	0.4409	0.4313	0.4521	0.4455	0.4223
Listnet	0.3801	0.3866	0.3952	0.3851	0.4008	0.4176
Adarank	0.4360	0.4345	0.4403	0.4538	0.4449	0.4209
MDPRank	0.4345	0.4421	0.4329	0.4509	0.4463	0.4315
DRLRank	0.4416	0.4452	0.4432	0.4598	0.4538	0.4365
Setrank	0.4383	0.4489	0.4486	0.4616	0.4503	0.4289
MA-DRLRank	0.4463	0.4379	0.4584	0.4526	0.4591	0.4388

Table 4.4: Results for different metrics on MSLR Web-30k Dataset

Method	NDCG@1	NDCG@3	NDCG@5	NDCG@7	NDCG@10	MAP
RankSVM	0.4361	0.4522	0.4607	0.4581	0.4543	0.4008
Ranknet	0.4380	0.4412	0.4523	0.4601	0.4585	0.4132
Listnet	0.4379	0.4366	0.4391	0.4414	0.4502	0.4176
Adarank	0.4451	0.4518	0.4611	0.4671	0.4579	0.4302
MDPRank	0.4411	0.4527	0.4623	0.4654	0.4611	0.4234
DRLRank	0.4507	0.4609	0.4507	0.4583	0.4641	0.4381
SetRank	0.4424	0.4587	0.4631	0.4508	0.4627	0.4331
MA-DRLRank	0.4587	0.4654	0.4579	0.4546	0.4707	0.4573

MSLR Web 30 k contains significantly more training data samples compared to the the other dataset. As a result, the deep deterministic policy gradient agent now gets more samples for training, which reduces variance and allows it to construct a more accurate model. We can again observe from the reward distributions of the two datasets that the algorithm is more stable on the MSLR Web 30k dataset. Again, because there are more samples in the dataset, we may generate a larger number of trajectories, which leads to a reduction in variance and hence higher stability in the training.

Furthermore, we can see that Listnet performs the worse on the two datasets when compared to other baselines. On the two datasets, we compared the performance of our method for varied discount factors, and the results are shown in Figure 4.3 and 4.4, respectively. Our approach performed best on the MSLR Web 30 k dataset with a discount factor of 0.99, while on the MSLR Web 10 k dataset with a discount rate of 0.95 and 0.99. The lowest reward was obtained using a discount factor of

0.70 for both datasets. In figure 4.6, we also compared our approach with varying numbers of agents. Consequently, we may conclude that a multi-agent framework for learning coordinated actions with shared observations and actions aids in learning a better model of the environment. Multiple agents share observations about the documents and learn a coordinated approach and coordinated actions for exploring the correlation between the documents. In figure 4.5, we also compared the reward distribution of our method to that of a single agent. On the two datasets, we can see that our approach converges faster and is more stable than on the single agent situation.