Chapter 3

A Deep Actor Critic Reinforcement Learning framework for Learning to rank

Reinforcement Learning based methods has been significantly applied in the ranking task with good success [41, 93]. Most of the existing RL-based methods for ranking utilize natural policy gradient methods [110, 115, 33, 69] however, the Policy Gradient based methods suffer from noisy gradients and high variance. The natural policy gradient algorithms like REINFORCE update the policy parameter through Monte Carlo updates. The random sampling from various stochastic policies can lead to high variability in log probabilities and cumulative reward values, as the trajectories might significantly deviate from each other during training [92]. As a result, it leads to noisy gradients and unstable learning. Even other techniques like providing a baseline function are not as effective as the state space is extremely large due to a large number of query document pairs. Reinforcement learning (RL) has proven to be effective in problems of different domains with non-differentiable metrics through policy gradient approach. For instance, successful results have been attained in summarization [74], machine translation [80], recommender systems [21], etc., using Reinforcement learning based methods. Bandit-based ranking models have also been proposed [77, 123, 89] based on implicit feedback, i.e., clicks for the query based ranking task. However, clicks are considered weak relevance signals because

they often are affected by noise and biases such as position bias. With position bias, higher ranked documents have more chance to be observed and therefore accumulate more clicks, even if they are not much relevant to the query[71, 46].

Due to the vast size of web search space, it is essential to find the most relevant documents for a specific query; hence ranking the search results is an important task in information retrieval. With large action space, i.e., a very large number of items, value-based methods in Reinforcement Learning are less effective as it becomes infeasible to calculate the value function estimates of all the actions in the state. We address the issue of Learning to Rank over large action spaces with Deep Reinforcement Learning using Policy gradient algorithms, which are known to be effective over large-scale datasets [31] as they directly learn optimal policy from the usage data. Further, the powerful function approximation properties of deep neural networks can be very effective in large action spaces. Deep Reinforcement Learning (DRL) has achieved significant success by scaling up complex problems which were earlier intractable with traditional RL methods due to the curse of dimensionality. DRL provides the powerful function approximation properties of deep neural networks and thus has been effectively applied in different complex problems with large action spaces(a large number of items), thus, combining the advantages of Deep Learning and Reinforcement Learning [77]. On the other hand, policy gradient algorithms learn the optimal policy directly by learning the policy parameters instead through neural networks. Policy-based approaches are useful for dealing with large action spaces, including continuous spaces with an infinite number of actions. The agent learns the policy directly and chooses an action from a probability distribution of the action space in policy-based techniques [94]. Also, for continuous actions, we can learn the parameters of a probability distribution and choose a value from there instead. Further, policy-based methods can learn stochastic policy, unlike value-based methods, and thus handle the explore/exploitation dilemma automatically.

When used in RL, the fundamental constraints of Deep Learning, such as instability and unpredictability, are exacerbated due to correlated and identically distributed

sample sets, estimation variances, and bias in function approximation [28]. Methods such as DDPG also update Q-value in a similar way to DQN, which makes it prone to overestimating Q-values for action, resulting in higher bias and instability in convergence. When updating the critic, deterministic policy techniques have a tendency to produce target values with a high variance which occurs due to overfitting to the skews in the value estimate [92, 130].

However, when the action space is high dimensional, as in a web search task with millions of documents, the variance in gradient estimation becomes large, leading to slow convergence of the algorithm and increase in sample complexity [107]. To address these issues, we propose a Deep Reinforcement Learning based methodology for learning to rank task by modeling the ranking process as a Markov Decision Process.

A general Learning to Rank problem is composed of N labeled queries. Each query is associated with a set of documents and the corresponding judgement labels with them. We formulate the problem as $\{V_i, X_i, Y_i\}_{i=1}^{N}$, where V_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. After the user submits a query, they are provided with the relevant list of documents. The goal in LTR, thus, is to rank the candidate documents according to their relevancy and provide the output list to the user.

3.1 Formulation of Learning to Rank as MDP

We modelled the learning to rank problem in a markov decision process framework as described in section 1.1. At each time step t, the agent receives the particular state s_t from the environment, and uses it select the action a_t following the policy ϕ_t. It then receives the reward r_t. Further, the agent moves to the next state s_{t+1}. We described the different components of MDP as:

State : The state space formation process is described as following:

1. The state space S, constitutes of set of states that describe the environment.
2. The state s_t at time t, as $[X_t, Y_t]$, where X_t represents the list of k documents and Y_t as the candidate set of all the items. While designing the state space we combine the vector list of k documents into a matrix.
3. We next feed the matrix using Convolution Neural Network layer [70] to extract the sequential patterns among the documents by capturing image features [96].
4. The output of the CNN module is then fed into two fully-connected ReLU network layers to acquire higher-level features, v_t.
5. We then compute the cross product with the previous state vector s_{t-1} and generate the current state s_t.

To begin with, the previous state vector s_0 is taken as a vector of all 1's. The state representation module is depicted in Fig. 5.1.

Action: We obtain the action from the state s_t in two stages for generating a ranked list from a continuous vector, which is the output of the actor network in the first step. Using the actor-network, we first produce a sub-action $a*$, that represents the intermediate action. We eventually get the action a_t, which is a ranked list of objects, from $a*$.

Sub-action Generation : To obtain the intermediate action from the present state s_t, we employ the deep neural network architecture. To get $a*$, we feed the s_t across several completely linked layers, as shown in Fig. 3.1. To obtain the intermediate action, $a*$, we fed the state through two layers with Relu activation functions and a final layer with Tanh activation. Then, as specified in Algorithm 5.5, we use $a*$ to construct the action, i.e., the ranking list of k documents.

Reward : The environment then provides feedback to the agent in the form of reward after generating the action, i.e. the list. The reward in the mdp formulation for the

ranking problem can be considered as an assessment of the relevance of the picked document to the query. The reward is then approximated as the difference between the list's DCG values at time steps t and t+1. For instance, let the documents be $D_1, D_2, ..., D_{10}$ and their corresponding relevance labels be $[1, 0, 2, 1, 0, 0, 0, 2, 1, 1]$. The DCG is then calculated as $\mathbf{DCG_p} = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)}$, where i denotes the index of the document, rel_i represents the label of the ith document. The reward function can directly optimise the IR evaluation measure using this approach, which has been shown to be effective in recent publications.

Transition : After receiving the reward, the agent advances to the next state $s_{t+1} = [a_t, D_t \backslash a_t]$. We utilise positive feedback for documents to move to the next state. The documents that the user finds relevant are appended to the next state, while the same number of documents are removed from the present state. For instance, let the present state s_t be document vector list $\{d_1, d_2, ..., d_{10}\}$, and the action a_t be $\{a_1, a_2, ...a_{10}\}$. If the documents relevant to the query are a_5 and a_7, then the next state s_{t+1}will be $\{a_5, a_7, d_3, ...d_{10}\}$. If no documents are found relevant, then the state remains unchanged.

Discount factor : The discount factor $\gamma \in [0, 1]$ represents the tradeoff between the immediate and future rewards. When γ is set to 0, the agent is only concerned with the immediate rewards and ignores the future rewards. Whereas, if γ is set to 1, the agent takes into account all the future rewards as well as the immediate rewards.

3.2 Algorithm

To address the issues of variance and instability described in Section ??, we propose an algorithm DRLRANK, based on the state-of-the-art algorithm twin-delayed DDPG (TD3) [37], to improve on these shortcomings by training the RL agent using techniques such as clipped double-q learning, delayed policy updates, and so on. It smooths the target policy using noise regularization to reduce variance. Furthermore, TD3 employs clipped double Q learning, in which the smallest value of

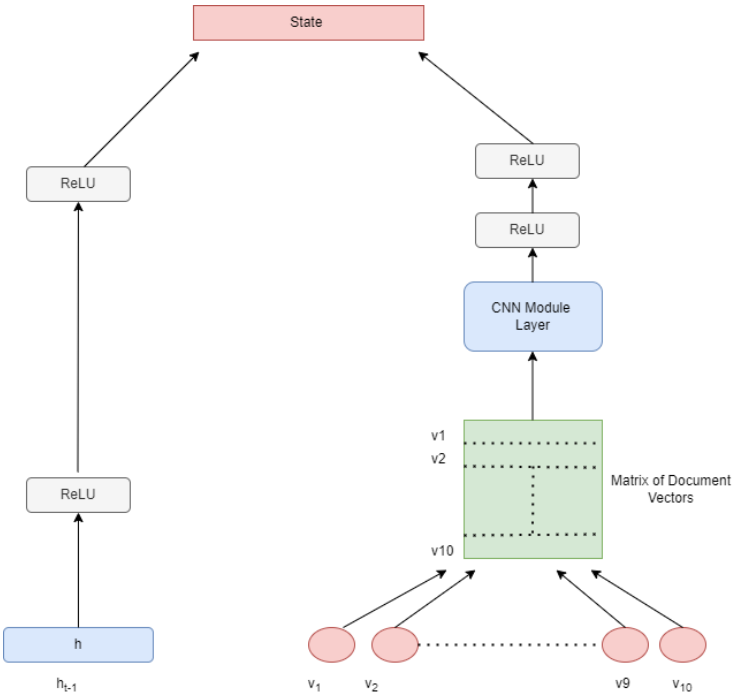


Figure 3.1: State Representation

the two critic networks is used to underestimate Q values. This, together with the delayed update, leads in a more stable approximation with reduced bias.

TD3 is an off-policy approach that maintains previous events in an experience replay buffer, then randomly samples transitions from it and feeds the sample data to actor and critic networks, enhancing sample utilization efficiency. In TD3, we discuss the actor-critic design that mitigates these concerns. The algorithm is depicted in Algorithm 2. The algorithm receives query document pairs as input and sets the policy and target parameters (Line 1-2). The algorithm obtains the environment's state and then employs the actor network to generate the sub-action (Line 4-5). It then receives the action, which is a list of ranked documents from Algorithm 1 (Line 6). The algorithm then provides the user the list, obtains the reward, and proceeds on to the next state (Line 7-9). The algorithm then saves the current state, the next state, the reward, and the action in the replay buffer (Line 10). The algorithm computes the target action and target Q value from the sampled transition using clipped double Q learning and then updates the Q-function using Gradient Descent (Line 11-14). Finally, policy and target networks are updated (Line 15-20).

Algorithm 1 Generating Action : Ranked Document list

1: Input : Intermediate Action a^*, List of Candidate Documents X_i
2: Output : Action List
3: // Generate the list of documents from the intermediate action
4: for each $x_i \in X$ do
5: compute $a^* * x_i$
6: end for
7: // select top k from line 5 as a_t
8: return a_t

Actor Network : The policy is defined by the actor. It receives the state as input and returns the action. The actor network is used to iteratively update its parameters ϕ and choose the current action a based on the current state s. We define the actor network as,

$$f_\phi : s \rightarrow a^* \tag{3.1}$$

We utilized a deep neural network framework to design the network as described in 4.2. A target network μ selects the optimal next action $a*$ according to the next state s' sampled in the replay buffer through a *clip* function . To generate the ranked list, we select the top k documents with the highest score, $a_t = \text{argmax}_i \, a^* * x_i$, where x_i denote the document vectors. The guassian noise is annexed to the action, which places the values within the range that the environment supports. This further adds the exploration of the environment.

$$a^* = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c)\right), \qquad \epsilon \sim \mathcal{N}(0, \sigma) \tag{3.2}$$

The target network parameters ϕ' are periodically updated from the actor parameters ϕ. The deterministic policy is used to optimize the parameters of the actor network, with the loss function is defined as,

$$\nabla_\phi J(\phi) = \sum_j Q_{\theta_1(s,a)}|_a \nabla_\phi \pi(\phi) \tag{3.3}$$

During the learning process, the target network employs a deep function approximator to provide a steady objective, resulting in improved convergence. However,

Algorithm 2 DRLRANK

Input: Query-Document Pairs $\{V_i, X_i, Y_i\}_{i=1}^{N}$, initial policy parameters ϕ, Q-function parameters θ_1, θ_2, empty replay buffer $\mathcal{H}$
Set target parameters equal to main parameters $\phi_{\text{targ}} \leftarrow \phi$, $\theta_{\text{targ},1} \leftarrow \theta_1$, $\theta_{\text{targ},2} \leftarrow \theta_2$
for t in range T do
    Observe the current state s
    Select action $a^* = \text{clip}(\mu_\phi(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$ //Generate the sub-action using Policy Network
    Generate the action, a, list of ranked documents from Algorithm 5.5
    Execute a in the environment //Present list to user
    Observe next state s', reward r
    Store (s, a, r, s') in replay buffer $\mathcal{H}$
    If s' is terminal, reset environment state.
    Randomly sample a batch of transitions, $M = \{(s, a, r, s')\}$ from $\mathcal{H}$
    Compute target actions

$$a'(s') = \text{clip}\left(\mu_{\phi_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right), \qquad \epsilon \sim \mathcal{N}(0, \sigma)$$

    Compute targets

$$w(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\theta_{\text{targ},i}}(s', a'(s'))$$

    Update Q-functions by one step of gradient descent using

$$\nabla_{\theta_i} \frac{1}{|M|} \sum_{(s,a,r,s',d) \in H} \left(Q_{\theta_i}(s, a) - w(r, s', d)\right)^2 \text{ for } i = 1, 2$$

    if t mod d then //Delayed Policy update
        Update policy by deterministic gradient ascent using

$$\nabla_\phi J(\phi) = \frac{1}{|M|} \sum_j Q_{\theta_1(s,a)}|_a \nabla_\phi \pi(\phi)$$

        Update target networks with
        $\theta_{\text{targ},i} \leftarrow \rho\theta_{\text{targ},i} + (1 - \rho)\theta_i$
        $\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi$
    end if
end for

errors in observed states might lead to suboptimal strategy. When an incorrect policy is overestimated, the learning diverges and exacerbates as it updates on states with errors. To minimise error propagation, the policy network must be changed less frequently than the value function. As a result, the policy network is required to be updated at a lower frequency than the value network. Less frequent policy modifications can result in less variance in Q-value function updates, resulting in a better policy.

Critic Network : The objective of critic network Q_{θ_i} is to predict the Q value based on the present state and action. The value function is used to offer feedback to the actor with regard to the present action. There are two critic networks and two critic target networks in the TD3 algorithm, which undergo updates less frequently than the critic models. Further, certain improvements are made by the TD3 algorithm to prevent overestimation of Q-values. It employs the concept of Double-Q learning, but with two critics instead of one to calculate the target q-value, w, ,

$$w_1 = r + \gamma Q_{\theta_2}(s', \mu_\phi(s')), \quad w_2 = r + \gamma Q_{\theta_1}(s', \mu_\phi(s')) \tag{3.4}$$

To avoid overestimation bias, the Clipped Double Q-learning objective utilizes the minimum of the two estimated Q-values.

$$w = r + \gamma \min_{i=1,2} Q_{\theta_i}(s', \mu_\phi(s')) \tag{3.5}$$

This underestimating bias is not a problem given that low values are not propagated through the algorithm as they are in the case of overestimation of Q values. This produces a more stable approximation, which enhances the algorithm's overall stability. Furthermore, to avoid overfitting, the calculation of Q-values must be smoothed in order to resolve the trade-off between bias and variance. As a result, a truncated normal distribution noise is applied to each action as a regularization,

resulting in the adjusted target update.

$$w = r + \gamma Q_{\theta'}(s', \mu_{\phi}(s') + \epsilon) \tag{3.6}$$

Consequently, the target policy smoothing mitigates the possibility of exploiting wrong actions if the Q-function approximator learns an erroneous sharp peak for actions, resulting in sub-optimal performance. The TD3 algorithm also delays updates to actor and critic target networks, updating them less frequently using Polyak Averaging approach, $\theta_{\text{targ},i} \leftarrow \rho\theta_{\text{targ},i} + (1-\rho)\theta_i$, $\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1-\rho)\phi$.

3.2.1 Significance of Each Component in Ranking Tasks

In the context of Deep Policy Gradient Methods with the Actor-Critic framework, the goal is to learn a policy that improves the ranking task, where the objective is to rank documents based on relevance to a given query. When documents are represented in vectorized form, such as in the LETOR dataset (which consists of labeled data for learning-to-rank tasks), a deep reinforcement learning (RL) approach like the Actor-Critic framework can be used to optimize the ranking process.

Actor: The actor represents the policy in reinforcement learning. It is responsible for selecting actions (in this case, the ranking of documents) based on a given state (a query or document feature vector). The actor outputs a probability distribution over the possible rankings of documents. The actor's objective is to learn a policy that assigns rankings to documents in response to a query. This can be treated as a sequential decision-making problem where each document's position in the ranking is an "action" taken by the actor. In the context of the datasets we utilized for experiments(LETOR), which provides feature vectors for each document and a relevance label, the actor learns to generate rankings by assigning positions to documents based on their feature vectors. The model aims to optimize the ranking in such a way that relevant documents (according to the relevance labels) are ranked higher.

Critic: The critic estimates the value of the current state-action pair (i.e., a given ranking of documents). This value function is used to provide feedback to the actor in terms of how good or bad its action (ranking) was. The critic evaluates the ranking produced by the actor by estimating the expected future reward or value of the current ranking. This value function serves as a guide for the actor to improve its policy. In ranking tasks, this evaluation can be based on a reward function that quantifies the quality of a ranking. We utilized commonly used evaluation metrics for ranking such as Normalized Discounted Cumulative Gain (NDCG), and Precision at k. The critic's value function can be tailored to estimate the reward associated with these metrics, helping the actor improve the ranking over time.

Objective Function and Its Improvement:

To optimize the ranking task, the objective function of the RL agent (actor-critic model) should be designed to improve the ranking quality iteratively. Here's how the different components contribute to this:

Policy Gradient (Actor): The policy gradient method updates the actor's parameters based on the gradient of the expected reward with respect to the actor's policy. Specifically, the objective is to maximize the expected reward by improving the policy such that relevant documents are ranked higher. In ranking tasks, the reward signal could be derived from ranking metrics like NDCG. For example, the gradient of the policy could push the model to rank highly relevant documents in the top positions.

The policy gradient objective can be formulated as:

$$J(\theta) = \mathbb{E}\left[\sum_{t} \gamma^t r_t\right]$$

where r_t is the reward at time step t (e.g., NDCG), and γ is the discount factor. By explicitly defining the reward function to reflect ranking performance, the actor can iteratively improve the policy to produce better rankings.

Value Function (Critic): The critic evaluates how good the current policy is by estimating the expected return (reward) for a given state-action pair. In the case of ranking, this can be based on the current ranking produced by the actor and its corresponding evaluation metric (e.g., NDCG). The critic's objective is to minimize the temporal difference (TD) error:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

where $V(s_t)$ is the estimated value of state s_t, and r_t is the reward for the current state-action pair (ranking). The TD error is used to adjust the critic's parameters, ensuring that the value function improves over time.

The critic helps the actor by providing feedback on which rankings are closer to the optimal ranking and thus guides the actor to improve the ranking policy.

Improving the Objective Function for Ranking:

- State Representation (Document Features): The state at each step is a representation of the query and its associated documents, typically in the form of feature vectors. The actor uses these feature vectors to generate a ranking. Deep neural networks can be used to process these document features to learn rich representations. This is crucial for improving the actors ability to make informed decisions based on the query-document pair.

- Actor-Critic Updates: The actor learns from the critics feedback. The critic evaluates the rankings using a value function (based on a ranking metric like NDCG), and the actor updates its policy accordingly. By optimizing the objective function with the gradient of the expected reward, the model iteratively improves its ranking policy. The critics value estimates also provide a better approximation of the ranking quality, further helping the actor refine its policy.

- Reward Function: We designed the reward function to capture the ranking quality utilizing the DCG metric directly as the reward function. For example,

if the top-ranked documents are highly relevant, the reward should be high, and the actor will be encouraged to rank these documents higher. Using ranking-specific rewards allows the actor to better optimize for ranking quality rather than merely predicting relevance scores.

3.3 Experiments

3.3.1 Evaluation datasets and experimental settings

We performed experiments to evaluate our algorithm's performance on the standard Learning to Rank datasets [76], OHSUMED, Million Query Track 2007 (MQ2007), and Million Query Track 2008 (MQ2008). The datasets are made up of a set of queries, a set of documents, and a set of relevance judgements. In datasets, the three possible relevance labels are relevant (2), slightly relevant (1), and non-relevant (0). OHSUMED and MQ2007 have 45 features, while MQ2008 has 46 features. We used the Letor standard features in all of the experiments, with each document represented as a vector. Table 4.1 provides statistics for the three datasets. A query-document pair is represented by a vector in each row of the dataset. For example,$query\_id, relevance\_label, feature-1, feature-2, ..., feature-45$, denotes a vector having a query id, related features, and a relevance label. The datasets' various characteristics include the document's TF-IDF, BM-25, Pagerank, URL stay duration, and so on.

We compared the results of our studies to the following state-of-the-art baselines: RankSVM [45], RankNet [12], AdaRank [117], A2C [66], DDPG [55], ACDRL [62], Setrank [73] and MDPRank [85]. We employed neural network-based implementations of Ranknet, Listnet, and Adarank. For the neural network, we used two hidden layers with 50 to 200 nodes each. The adam optimizer was used to train the distinct baselines based on the ranking loss function of each model. The individual hyperparameters for the various baselines are picked using a grid search from the values specified in Table 4.2. For RankSVM method, we used a linear kernel with a

regularization rate 10^{-3}.

We trained the RL agent for 1000 episodes using a learning rate of 10^{-3} and a discount rate of 0.99 for the MDPRANK algorithm. We utilised a learning rate of 10^{-3} and a discount rate of 0.99 for A2C and DDPG. In addition, for the DDPG method, we used a batch size of 128 and a buffer size of 10000. For our method, we utilized a learning rate of 10^{-3} for the actor and 10^{-4} for the critic, as well as a discount rate of 0.99. We utilised the Adam optimizer to set the network parameters and set a learning rate of 10^{-4} for the actor-network and 10^{-5} for the critic. We utilised the standard parameters stated in the paper for the ACDRL approach.

In all of the trials, we conducted 5-fold cross-validation on these datasets and provided the average of the five folds findings. Our trials were performed out on an Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz processor,x86_64 architecture, with 264 GB RAM.

Evaluation measures : To evaluate the performance of the proposed algorithm on the datasets, we use two common IR assessment measures, Normalised Discount Cumulative Gain (NDCG) [43] and Mean Average Precision (MAP) [44] . We evaluated NDCG at positions 1, 3, 5, and 10 in the experiments.

3.4 Experimental Results

The experimental results for the OHSUMED, MQ2007, and MQ2008 are shown in Tables 4.3, Table 4.4 and Table 3.5, respectively. NDCG and MAP are two different evaluation measures. The NDCG measure was examined at positions 1, 3, 5, and 10. Except for NDCG@5, our approach outperforms the different baselines for most ndcg measures in the OHSUMED dataset. For OHSUMED dataset, our approach significantly outperforms the baselines for NDCG@1 and NDCD@5. Similarly, our approach outperforms other baselines on the MQ2008 dataset, with the exception of NDCG@3 metric. Overall, our technique improves over the MQ2007 dataset the most among the three datasets. We hypothesise that the reason for this improve-

ment among the three datasets is because MQ2007 has the most training documents and queries, and thus the Deep RL agent can learn a more complex model than the other two. Furthermore, when compared to the OHSUMED dataset, the performance of various baselines, including our approach, is more stable in MQ2007 and MQ2008. The query space of the three datasets can also be used to analyse this behaviour; while MQ2007 has 1700 inquiries and MQ2008 has roughly 800 questions, the OHSUMED dataset has only 100 queries. Because there is more sample data available, the policy gradient technique can develop a better model, resulting in lower variance and more stable learning with gradient descent. Furthermore, we can see that the Adarank method performs the worst on OHSUMED when compared to other datasets, while Ranknet performs the worst on MQ2007. Figure 3.2 depicts the reward distribution of our algorithm across multiple datasets and episodes. As the number of episodes increases, we can see that our algorithm, DRLRank, is able to attain steady learning for the three separate datasets. Furthermore, we can see from the reward distribution over the episodes that our algorithm achieves greater stability on MQ2007. We tested our algorithm's performance with varied discount factors and provided results for the MQ2007 dataset in Figure 3.3.

Table 3.1: Statistics on OHSUMED, MQ2007, MQ2008 Datasets

Dataset	Queries	Documents	Features	RelevanceLabels
OHSUMED	106	16,140	45	3
MQ2007	1692	69,623	46	3
MQ2008	784	9360	46	3

Table 3.2: 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	32	64	128	256

In Figure 4.6, we also demonstrated the performance of our method over the NDCG measure with various discount factor. We can see from the results that our algorithm provides optimal reward and NDCG performance across all datasets with a discount

Table 3.3: Results for different metrics on OHSUMED Dataset

Method	NDCG@1	NDCG@3	NDCG@5	NDCG@10	MAP
RankSVM	0.4921	0.4711	0.4423	0.4554	0.4321
Ranknet	0.4803	0.4601	0.4344	0.4392	0.4223
Listnet	0.5082	0.4754	0.4361	0.4488	0.4176
Adarank	0.4810	0.4054	0.4392	0.4286	0.4209
ACDRL	0.4921	0.4622	0.4215	0.4302	0.4212
A2C	0.4764	0.4180	0.4229	0.4264	0.4166
DDPG	0.5011	0.4716	0.4383	0.4608	0.4281
SetRank	0.5113	0.4665	0.4421	0.4567	0.4361
MDPRank	0.5108	0.4633	0.4322	0.4424	0.4387
DRLRANK	0.5273	0.4782	0.4379	0.4626	0.4406

Table 3.4: Results for different metrics on MQ2007 dataset

Method	NDCG@1	NDCG@3	NDCG@5	NDCG@10	MAP
RankSVM	0.3956	0.3982	0.4057	0.4124	0.4237
Ranknet	0.3884	0.3902	0.3938	0.4131	0.4188
Listnet	0.4002	0.4072	0.4143	0.4238	0.4214
Adarank	0.3877	0.3954	0.3921	0.4126	0.4221
ACDRL	0.4023	0.4116	0.3910	0.4112	0.4267
A2C	0.3961	0.4042	0.4130	0.4258	0.4287
DDPG	0.4170	0.4118	0.4252	0.4295	0.4277
SetRank	0.4117	0.4308	0.4268	0.4264	0.4186
MDPRank	0.4152	0.4164	0.4218	0.4224	0.4203
DRLRANK	0.4206	0.4243	0.4310	0.4353	0.4388

Table 3.5: Results for different metrics on MQ2008 Dataset

Method	NDCG@1	NDCG@3	NDCG@5	NDCG@10	MAP
RankSVM	0.3686	0.4198	0.4550	0.2268	0.4464
Ranknet	0.3341	0.3902	0.4321	0.2172	0.4408
Listnet	0.3802	0.4409	0.4612	0.2134	0.4505
Adarank	0.3878	0.4434	0.4781	0.2405	0.4402
ACDRL	0.3827	0.4319	0.4484	0.2465	0.4508
A2C	0.3811	0.4283	0.4676	0.2101	0.4449
DDPG	0.3774	0.4378	0.4723	0.2551	0.4391
SetRank	0.3928	0.4468	0.4758	0.2798	0.4166
MDPRank	0.3846	0.4354	0.4714	0.2641	0.4481
DRLRANK	0.3984	0.4417	0.4682	0.2831	0.4706

factor of 0.99. We used significance testing (two-tailed Student's t-test) [96] with p-value 0.05 to compare the statistical significance of our method's results across different baselines. The results were statistically significant for the vast majority

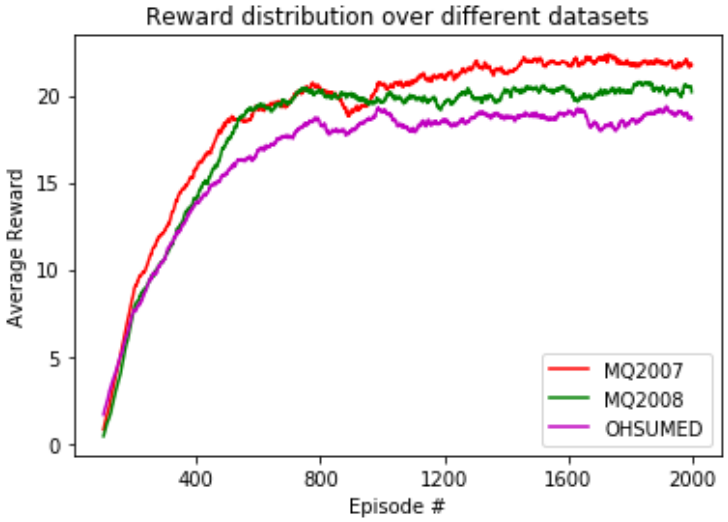


Figure 3.2: Reward Distribution

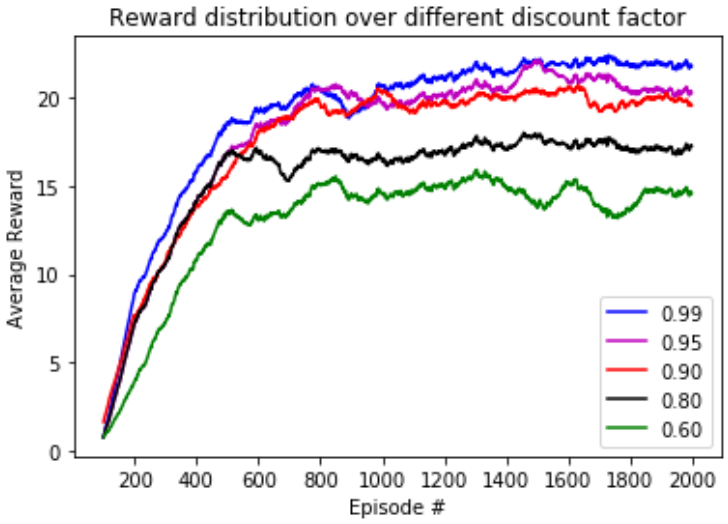


Figure 3.3: Varying Discount factor

of metrics across all datasets (denoted by † in Tables 4, 5, and 6). In 3.5 and 3.6, we examined the effect of various parameters on the NDCG metric, such as batch size and episode length. We can observe that the model's performance gradually increases with the batch size. The model produces the best results for NDCG@1 with batch size of 128 for MQ2007 and OHSUMED. Furthermore, for MQ2008, batch size 64 yields the optimal results. However, for the OHSUMED and MQ2008 datasets, we can see that performance begins to decline slightly after batch size of 128. One possible explanation is that the reduced ability to generalzse at larger batch sizes leads to lower performance.

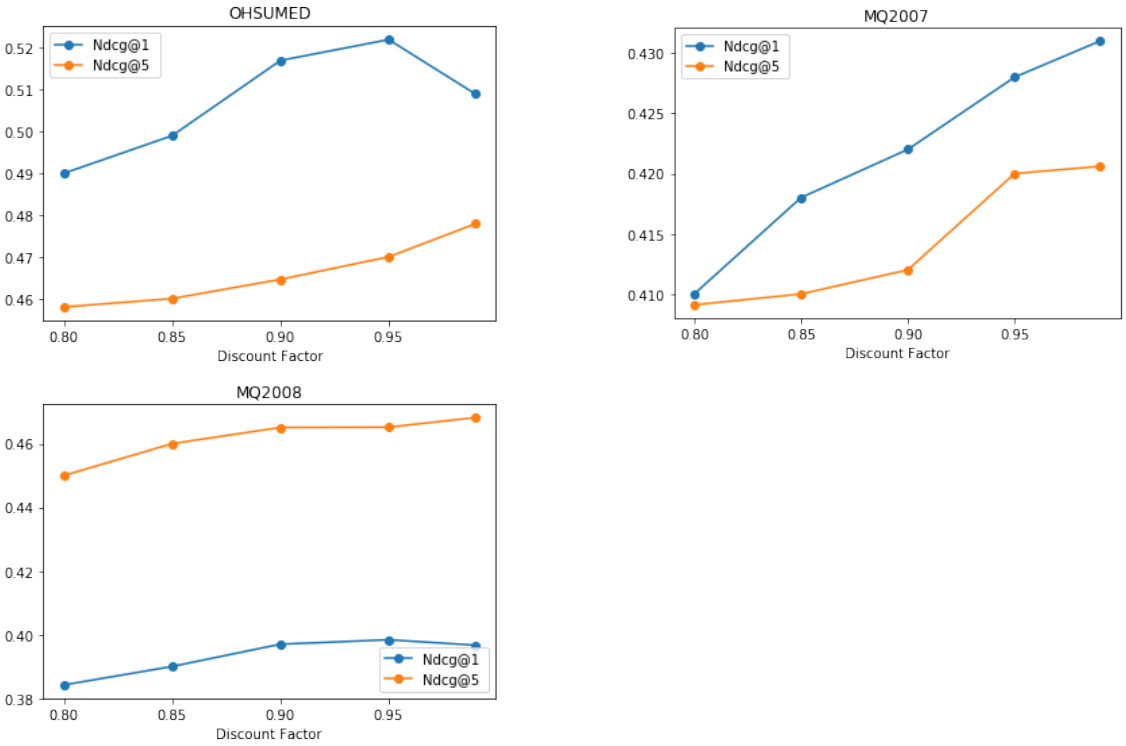
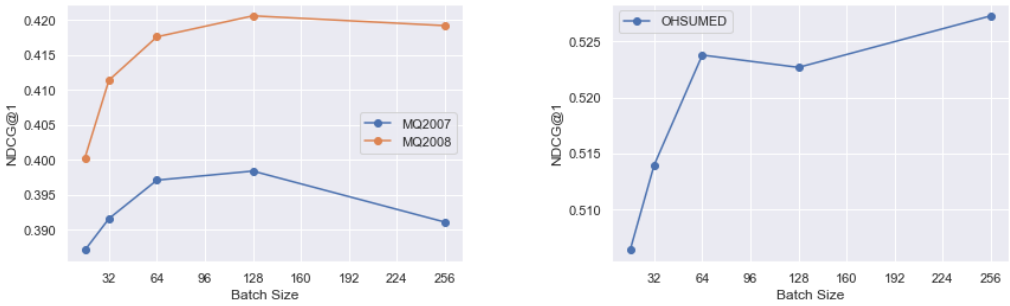


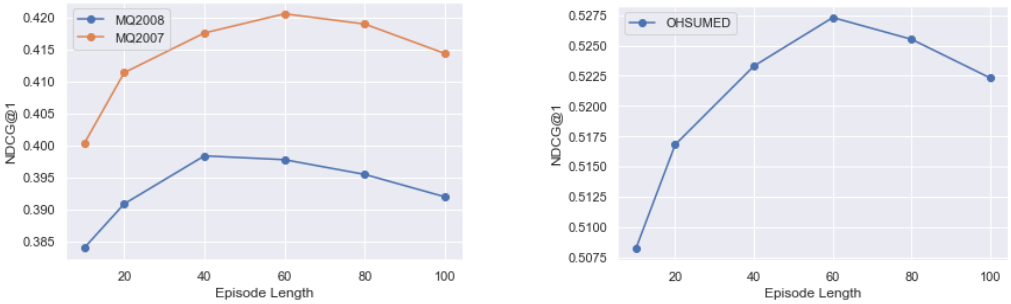
Figure 3.4: NDCG metric with different discount factor, x-axis: Discount factor, y-axis: NDCG

Fig 3.6 depicts how the episode length affects the performance of the model. We can see that intially, performace increases with the increase in episode length on various datasets. We can infer that when the episode time is too short, the agent does not fully interact with the environment and does not acquire enough experience for learning. We can also see that our approach outperforms the others on the MQ2007 dataset for a wide range of episode lengths. On the other side, we can see that performance drops after a specific episode length, with the intermediate range



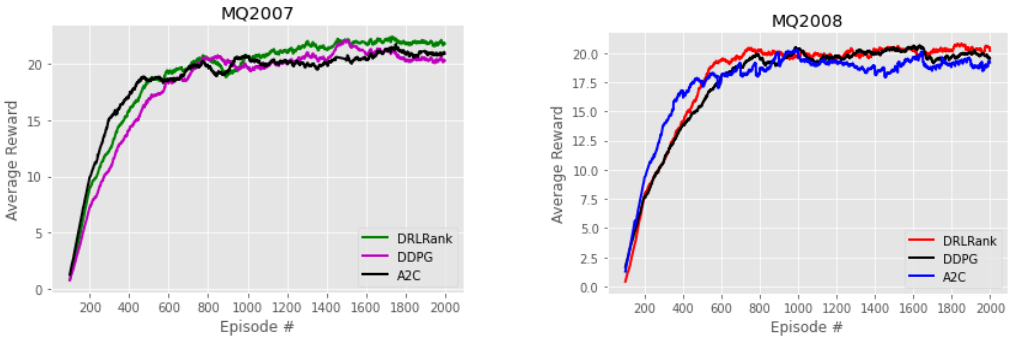
(a) (b)

Figure 3.5: Varying Batch Size



(a) (b)

Figure 3.6: Varying Episode Length



(a) (b)

Figure 3.7: Comparing DRLRank with other DRL algorithms

producing the best results across all datasets. We can speculate that the reason is exploration-explotation trade-off and thus optimal episode length is required to maintain a balance of both. In 3.7, we compared the performance of our algorithm to that of other DRL algorithms, DDPG and A2C. It can be observed that DRLRank converges better than the other two, while DDPG performs relatively better than A2C. One of the possible explanations is the overestimation bias of Q values in DDPG. This occurs when the agent's estimated Q values are greater than the real values, causing the RL algorithm to overestimate future rewards. As a result, the policy may incorrectly exploit the overestimated values and suffer from erroneous learning. To address the overstimas bias, we use clipped q learning and target policy smoothing techniques in DRLRank.

We can infer from the results that the DRLRank algorithm performs well on LTR problems, and that the more query-documents available for the policy gradient approach, the better the performance. This reinforces the concept that policy-based

techniques are more effective in large data sets since the algorithm has access to more sample trajectories, reducing the variance of the experiments. We also believe that the Deep RL technique can be applied to larger scale datasets. We attribute the algorithm's improvement to the Deep Reinforcement Learning framework, which can learn complex function of the environment/model in high-dimensional action space.