Deep Q Learning based Recommender Systems

Aditi Kanaujia
Language Technology Institute
Carnegie Mellon University
Pittsburgh, PA 15213
akanauji@andrew.cmu.edu

Anirudh Kannan
Language Technology Institute
Carnegie Mellon University
Pittsburgh, PA 15213
akannan3@andrew.cmu.edu

Pragnya Sridhar
Language Technology Institute
Carnegie Mellon University
Pittsburgh, PA 15213
pragnyas@andrew.cmu.edu

Sathyanarayanan Ramamoorthy
Language Technology Institute
Carnegie Mellon University
Pittsburgh, PA 15213
sramamoo@andrew.cmu.edu

Abstract

Today, recommendation systems are more prevalent than ever. While collaborative filtering was once an efficient approach, it is no longer robust given the volume of data and interactions that are required to be modeled today. Recent works have achieved significant success by using Reinforcement Learning agents in the context of recommender systems. Existing literature has shown the effectiveness of using Deep Q Networks to aid the Reinforcement Learning agent's decision making. To this end, we explore two formulations of Deep Q Networks in a Reinforcement Learning setting namely CDAS-DDQN and Naive-DDQN to provide recommendations. Experimental results evaluated on the MovieLens dataset highlight the viability of our approaches. For implementations, refer to this GitHub page.

1 Introduction

Recommender systems are used to model user behavior in order to suggest new content/items to users as a means to improve interaction. Collaborative filtering was one of the earliest techniques used in recommender systems. In this approach, matrix factorization of the latent space vectors (of users and items) was used to model user behaviour. However, when user, item, and associated data interaction increases, it becomes more challenging for basic approaches like collaborative filtering to capture this dynamic relationship.

Reinforcement Learning is a trial and error game where favorable actions are rewarded (positive reinforcement) and unfavorable actions are penalized (negative reinforcement). The system involves an environment which represents the rules in that system, multiple states of this environment and an agent who will decide using a value and take an action in order to transition between states of the environment. These models especially come in handy when dealing with infinite search space of environment states such as self driving cars or chess games. In order to enhance the degree of interaction being captured, we intend to apply reinforcement learning to the recommender system problem in this study.

An agent can select whether to pursue an action from a state using the Q-values associated with each action that are learned using the model-free technique of Q Learning. The expected payoff for each action taken from a certain environmental condition is determined using the Q table. Then, agents can act in a way that optimizes the payoff. The Bellman equation is used to create the Q table. This Q table is supplanted with a neural network in deep learning, which forecasts the Q values for each action based on the present state. The goal of Deep Q Learning is to learn the optimal Q function i.e the function that takes state inputs and outputs an estimate of future rewards, especially when the action and state spaces are of huge scale.

Double DQN (DDQN) is a deep Q-network algorithm variant that was developed to alleviate some of the issues that standard DQN might create, such as overestimating the value of some actions [van Hasselt et al. [2015]]. During the learning process of double DQN, the agent employs two different neural networks: a "main" network for choosing actions and a "target" network for assessing the predicted values of those actions. As a result, the agent is able to separate the decision-making process for actions from the evaluation of potential long-term rewards. This can aid avoiding overly optimistic value estimations and enhance the stability of the learning process. In this work, we empirically highlight the comparative performance of DDQN models in the context of recommender systems. We contrast two different approaches that differ in their formulation of action spaces and verify the viability of using DDQNs.

2 Literature Review

With the rise in the usage of technology across decades, it was natural for people to take advise from machines while making decisions, especially in trivial day-to-day tasks like shopping, browsing, etc. This led to a class of systems known as Recommender systems (RS). Goldberg et al. [1992], Resnick and Varian [1997] are one of the earliest works on Collaborative filtering and RSs. RSs are not directly collaborative systems but are used to suggest items based on users' interests. Content-based filtering Pazzani and Billsus [2007], Lops et al. [2011] introduced RS approaches that followed a more structured approach while recommending based on user needs. Both the above approaches suffer from limitations like sparsity, scalability, inability to adapt to new users, and poor recommendations (Ricci et al. [2011], Bobadilla et al. [2013]). Merged RS Su and Khoshgoftaar [2009] that incorporated both properties could not provide successful results. Recently, machine learning, especially deep learning techniques Goodfellow et al. [2016] are being used for various real-world tasks. They have the ability to adapt to extremely complex tasks and identify relations. Reinforcement learning is one such semi-supervised technique that learns patterns by interacting with the environment.

Reinforcement Learning based Recommender Systems can be broadly classified into RL and Deep RL (DRL) algorithms. RL based algorithms do not employ deep learning for policy optimization. State representations for these algorithms can either be using items as states (for instance, WebWatcher Joachims et al. [1996]) or using features from items, users, etc. as states. In the absence of deep learning, RL-based algorithms bank on the fact that a small subset of items that the user has already interacted with is good enough to support policy optimization (Taghipour et al. [2007]) using N-gram and sliding window over the last 'k' visited pages for instance). The same concept can be applied when extracted features are being used as states, like in Mahmood and Ricci [2007], Mahmood and Ricci [2009]. For Policy Optimisation, the most common algorithms include Q-learning and Sarsa because of their simplicity. Some papers employing this include Choi et al. [2018]; Chang et al. [2021]. Approximate methods for optimization, such as fitted Q (Lu and Yang [2016] etc) and gradient value iteration (Zhang et al. [2017]) are also used for policy optimization. The reward formulation may be a sparse numerical function (R1) or a function of observations from the target environment (R2). With RL based algorithms, R2 shows more promise than R1 with a 6:4 usage ratio. For environment building, RL based algorithms go for offline methods, although simulation is a fair method as well. DLR based algorithms on the other hand, use deep learning for policy optimisation. The state representation is dense, low-dimensional vectors called embeddings, which can be further encoded using RNN (Zhao et al. [2018]). This work further introduces the concept of having positive and negative states. Only about 20% of DRNs use extracted user and item features as state representations, for instance DRN for new recommendation by Zheng et al. [2018]. Policy optimisation for DLR can be divided into value based, policy gradient and actor-critic method. Value based methods include DQN and its extensions, with three main elements consisting of Q network architecture, experience replay and exploration. Policy Gradient methods do not need value functions, for instance, REINFORCE method. Lastly, Actor critic methods employ DDPG, combining DPG and DQN, for instance, Wolpertinger in Dulac-Arnold et al. [2015]. Reward formulation is primarily R2, with some works exploring approximate regretted awards like Robust DQN in Chen et al. [2018].

3 Related Works

3.1 Naive Collaborative Filtering

Vanilla Collaborative Filtering is based on similarity matrices calculated between User-User, Item-Item or User-Item. These similarity matrices are then used to predict the top-k recommendations for any given user. In this set of experiments, we recreated a User-User Similarity model to evaluate precision and recall when run on the 1M MovieLens dataset Harper and Konstan [2015] . Mathematically, let M be movies common to the two users, User 1 and User 2, being considered at any given time. If M_1 is the set of User 1's movies and M_2 the set of User 2's movies, then,

$$S = \frac{M}{\sqrt{M_1} * \sqrt{M_2}} \tag{1}$$

Where S is the similarity between User 1 and User 2. A prediction score is then calculated for each new (unrated) movie in the dataset with respect to each user and this prediction score is used to rank top-k recommended movies for that user.

The experiment is run to calculate the Precision and Recall on Top-5, Top-10 and Top-20 movie recommendations on the 1M MovieLens dataset.

3.2 Neural Collaborative Filtering

The matrix factorization method of collaborative filtering revolves around the inner product between latent feature vectors of users and items. This according to He et al. [2017] can be replaced by neural networks to learn more implicit patterns. Thus He et al. [2017] aims to observe the performance enhancement of deep neural network enabled collaborative filtering on the task at hand. In order to do this, three models were introduced - Generalized Matrix Factorization (GMF), Multi Layer Perceptron (MLP) Collaborative Filtering, and an ensemble of the two called Neural Matrix Factorization (NeuMF). User (p_u) and Item (q_i) Latent Feature Embedding vectors are obtained from a Fully Connected Embedding layer after which, they are fed into the deep neural networks to map them to prediction scores.

GMF scales the element-wise product of the user and item latent vectors with edge weights h and encapsulates this affine value within an activation function a_{out}. From this is can be observed that Matrix Factorization is special case of GMF where a_{out} is identity and h is unit vector:

$$\hat{y}_{ui} = a_{out}(h^T(p_u \odot q_i)) \tag{2}$$

Instead of element-wise multiplication, the MLP approach uses a concatenation of p_u and q_i. Since simple concatenation cannot capture the interaction between the vectors, hidden layers are introduced to pick up patterns. From experimentation, the authors found three Layer MLP to work best and chose ReLU activation, as it encourages sparse activations inorder to avoiding overfitting. As for the structure, the layer size halves on going deeper into the network.

$$z_1 = ReLU(h_1^T([p_u, q_i])) \tag{3}$$

Finally the NeuMF approach is an ensemble of the GMF which applies a linear kernel and MLP model which applies a non linear kernel. It is fine tuned by training over the pretrained weights of its components to observe significant improvement.

On evaluating, He et al. [2017] finds that all the three Neural Collaborative Filtering methods outperformed other methods like ItemPop, ItemKNN and BPR.

4 Proposed Framework

4.1 Formulation

Typically, a Reinforcement Learning (RL) problem is represented as a Markov Decision Process (Afsar et al. [2021]). A Markov Process is a sequence of random states where, the current state of a Markov system depends only on its immediate previous state.

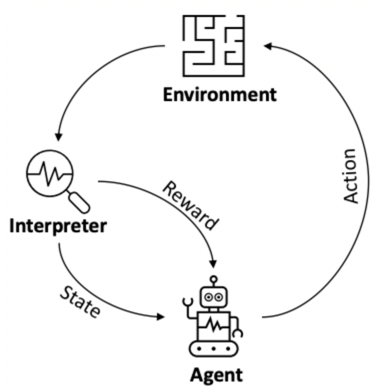

Figure 1: Agent-Environment Reinforcement Learning Paradigm Afsar et al. [2021]

As shown in Figure 1, the most popular approach to model a RL problem is by using the agent-environment interface. At every time step, the agent gets information from the environment in the form of a state. Based on this state information and the policy, the agent takes an action. The environment provides a numeric reward as feedback to the agent.

For a recommender systems, we define the Markov Decision Tuple (S, A, R, P, γ) as follows:

- **State**: A State $s_t \in S$ is an abstraction that represents the history of interactions the user has had with the system as well as user meta data. We use the **SR3** Afsar et al. [2021] i.e. encoded embeddings for state representation. The user, item and context features are converted into low dimensional embeddings and are used as state representations.
- **Action** : Action at a given timestep t, $a_t \in A$, is the item recommended to the user.
- **Reward** : Reward at timestep t, $r_t(s_t a_t) \in R$ is the reward received based on the user feedback for the recommendation. The reward function we propose to use is **R1** Afsar et al. [2021], because the labels we use are numeric ratings.
- **Transition Probability** : It is defined as the probability of transitioning from state s_t to state s_{t+1} if the agent successfully takes the action.
- **Discount Factor** : The discount factor $\gamma \in [0, 1]$ is used to weigh the rewards since the problem becomes more stochastic as the number of future timesteps considered increases. The discount factor enables the agent to factor in both long term and short term reward optimization.

4.2 Q Learning

Q learning is a Reinforcement Learning algorithm that heuristically chooses the next best action given the current state. In order to learn, the algorithm chooses the action randomly with the aim to maximize its reward. In essence it solves the Bellman Equations by maintaining a Q-Table $Q(a, s)$ that stores how good taking an action a from that particular state s is. The learning can be described as:

$$new_Q(s, a) = Q(s, a) + \alpha \left(R(s, a) + \gamma Max_a Q'(s', a') - Q(s, a)\right) \tag{4}$$

Where $R(s, a)$ is the reward gained for taking action a from state s, α is the learning rate, γ is the discount rate, $Max_a Q'(s', a')$ is the maximum expected q value from the next state.

This algorithm however does not scale as the number of states and actions increase as it requires huge amount of memory for the table and computation to compute the values.

4.2.1 Deep Q Learning

Considering the limitations of Q-learning based methods, Deep Q Learning uses Deep Neural Networks instead of a table to map input states to $(action, value)$ pairs. The input to the networks

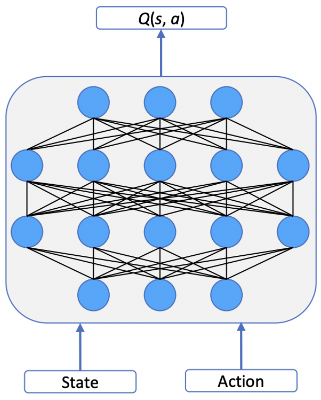

Figure 2: Abstract representation of a Deep Q Network Afsar et al. [2021]

are the states, and the outputs are an $(action, q - value)$ pair. Thus this network is responsible for action-value approximation. The DQN, as shown in Figure 2 uses experience replay to update weights during the training phase. Replay enables it to track experiences over time.

The loss function is therefore defined as

$$L_i(\theta_i) = \mathbb{E}_{s,a\text{-}p(.)}[(y_i - Q(s, a; \theta_i))^2] \tag{5}$$

where y_i is the target for iteration i.

4.3 Methodology

In our study, we experiment with two approaches, the **Naive DQN** and **Constrained Dynamic Action Space DQN**, primarily differing with respect to their action space formulations. In the Constrained Dynamic Action Space approach, at any given state, only a subset of actions is available to the Agent. Below we expand on the methodologies followed in both approaches.

4.3.1 Naive DDQN

DQNs generally work on fixed action spaces i.e given a state, the actions the agent can take are fixed. To incorporate this the implementation was narrowed down to the following structure :

- **State Representation**: The state of an agent is represented by a joint embedding of the user identifier and their top-rated movies. Embeddings are created for the user and the movie list separately followed by concatenation.
- **Action Space**: The appropriate action is to recommend a new movie. This leads to the transitioning of the agent from current state to next state
- **Reward Function**: User ratings are considered as rewards, thereby letting the agent improve based on the received feeback. Further, multiple formulations of the same were explored including zero centering them to penalize the model, using a binary reward system where a movie is given a positive score if the user has watched it and rated it highly. Unwatched movies are assigned 0 reward.
- **Transition Probability** - When the user watches the movie the agent recommends to the user they move into a new state. This new state is defined by the user and the new set of 5 movies for this user is made by removing the oldest movie in the previous state (the first movie to be added to the state, at index 0) and appending the newly watched movie to the movie list. Embeddings are obtained similar to States.

4.3.2 Constrained Dynamic Action Space DDQN

In this approach, we consider a small action space that is dynamically sampled at a given timestep to enable the agent to deal with large action space sizes as well as changes to the action space.

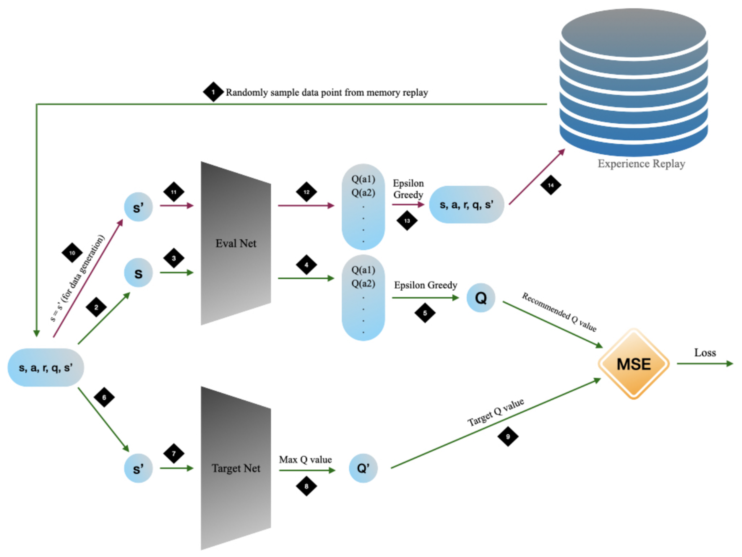

Figure 3: Full Action Space Approach

Additionally, we only consider movies that have been rated highly by the users in the training set. An overview of Constrained Dynamic Action Space DDQN (CDAS-DDQN) this approach can be seen in Fig 4

- **State Representation**: The state of an agent is represented by a joint embedding of the user identifier and a randomly sampled history set of five movies that the user has watched and rated highly.
- **Action Space**: The action space consists of movie embeddings that can be recommended to the user in the next timestep. Since the action space is very large, we postulated that making our network output Q values for all potential actions would be inefficient in terms of training time and learning. Hence, we constrain our action space to a set of 20 randomly sampled movies. To achieve this, we maintain two sets, **current action space** and **candidate action space**. The current action space consists of 20 randomly sampled movies and the candidate action space consists of all the remaining movies watched by the user.
- **Transition Probability**: In a single iteration the Agent selects the action with the largest Q value by passing the state and action embeddings to the DQN. Once an action is selected, we update the state similar to autoregression, by removing a movie from the movie history and adding the predicted movie. Furthermore, to avoid repetition in the movie recommendation, we replace the recommended action from the current action space with an action sampled from the candidate action space.
- **Replay Memory**: A replay memory buffer is used to store experiences including the current state, transitioned state, reward received and current action space. After every state transition, the experience will be stored in the replay memory. Once the replay memory is full, it is used to train both the evaluation and target Deep Q Networks and the replay memory buffer is then cleared.

5 Experiments

5.1 Dataset Description

We use the MovieLens 1M dataset Harper and Konstan [2015] to establish baselines for different models. The MovieLens dataset consists of 1,000,209 anonymous ratings for around 3,900 films submitted by 6,040 individuals who joined MovieLens.

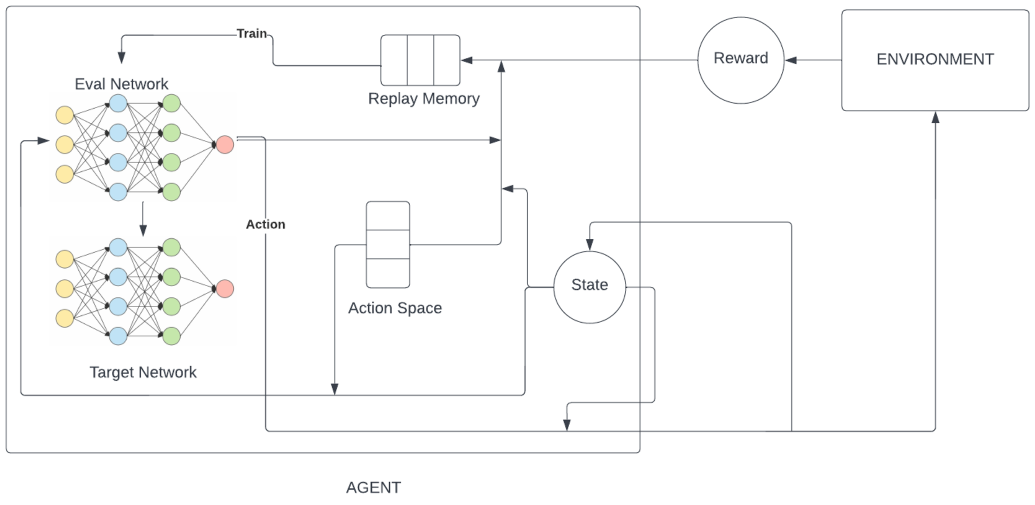

Figure 4: Constrained Dynamic Action Space Approach (CDAS-DDQN)

5.2 Training Baselines

A description of baselines used and their setup:

- **SASRec** : The SASRec Kang and McAuley [2018] model was trained for 600 epochs with a learning rate of 0.001, 50 hidden units and a maxlen of 200.
- **LightGCN** : LightGCN (He et al. [2020])was trained for 50 epochs with a batchsize of 1024 with a learning rate of 5e-3.
- **GMF**: The GMF was trained using an embedding size of 64, for 40 epochs with learning rate 0.001. A batch size of 256 and Adam optimizer was employed for quick convergence with accordance He et al. [2018].
- **MLP**: 3 Layer MLP with layers 64 (input layer), 32, 16, 8 was used in reproducing results. The model was trained for 40 epochs on embedding vectors p_u and $q_i$of size 32 each, with 0.001 learning rate, Adam optimizer and batch size 256 as mentioned in He et al. [2018].
- **NeuMF**: This ensemble model from He et al. [2018] was also trained for 40 epochs with learning rate 0.001 and batch size 256. When training with pretrained weights, the model was trained using SGD optimizer and when trained from scratch Adam Optimizer was used.

5.3 Evaluation Metrics

We report the normalized Discounted Cumulative Gain (nDCG) as our performance metric computed over top-5, top-10 and top-20 recommendations. We use nDCG as our preferred metric (over others like Mean Average Precision) because it considers both the relevance of retrieval as well as the ranking of items. This is a common metric used to measure the relevance of web search results. The formulation for NDCG score is -

$$DCG_p = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{log_2(i+1)} \tag{6}$$

$$IDCG_p = \sum_{i=1}^{|REL_p|} \frac{2^{rel_i} - 1}{log_2(i+1)} \tag{7}$$

$$nDCG_p = \frac{DCG_p}{IDCG_p} \tag{8}$$

Table 1: Results obtained for implemented baseline models obtained on MovieLens 1M Dataset. nDCG scores are reported

Model Name	**nDCG @ 5**	**nDCG @ 10**	**nDCG @ 20**
SASRec	0.5261	0.5629	0.5859
Generalized Matrix Factorization	0.3690	0.4231	0.4621
Multi Layer Perceptron	0.3424	0.4008	0.4383
Neural Matrix Factorization (with Pretraining)	0.3767	0.4316	0.4686
Neural Matrix Factorization (w/o Pretraining)	0.3696	0.4226	0.4579
LightGCN	0.4358	0.4046	0.3871
CDAS-DDQN	**0.1303**	**0.1798**	**0.2281**
Naive-DDQN	0.0908	0.1355	0.2041

5.4 Training CDAS-DDQN

For training the CDAS-DDQN, we first synthesize the start state for each user by passing the user identifier along with 5 randomly sampled movies that have been rated through an embedding layer. The initial action space for each user was filled with 20 movies from the list of rated movies, and the remaining movies were stored in a candidate action space for replacement in the future. Mean Square Error (MSE) Loss was used to train the DDQNs. For each episode, the agent was trained for either 10 or 20 timesteps (depending on the ablation configuration) and the experiences were stored in the replay memory. Once the replay memory was full, the examples from the replay memory was used to train the eval net and target net. A learning rate of $1e^{-3}$, gamma value of 0.9, epsilon value of 0.9 along with batch size of 5 was used.

5.5 Training Naive-DDQN

Our implementation of DQN consists of three Convolutional Layers (1D) followed by 2 Linear layers. In addition to this, after every layer, batch norm, dropout and activations have been implemented. Each user has a different start state and in order to handle that, we initialize the experience replay memory with the iinitialstates of all users prior to iterations. We use MSE for our loss computation, AdamW to ensure weights decay in the neural network, and ReduceLROnPlateau scheduler for the learning rate initialized to $1e^{-3}$. The training algorithm described in Mnih et al. [2013] has been implemented. This process can be understood from Fig. 3

6 Results

In case of Naive-DDQN, from the nDCG @ 20 values of Table 6.1, we observe that increasing the exploration is generally good for performance. However, this needs to be rigorously tested with a network of sufficient capacity and for a large number of epochs. We also find that zero centering helps the model learn better. However, owing to discrepancy in this pattern for nDCG @ 5 and nDCG @ 10, the model needs to be run for longer number of iterations to come to concrete conclusions.

We also observe CDAS-DDQN performs significantly better than the Naive-DDQN approach. It can be observed that models with smaller embedding sizes seem to perform better. Additionally, training the model for larger number of timesteps helps improve the nDCG values. One of the main advantages of the proposed CDAS-DDQN approach is that the model in the Naive DDQN uses a static action space and hence requires all the movies to be known beforehand.

6.1 Comparison with Baselines

Among our approaches, CDAS-DDQN emerging as the best model achieving the highest value of 0.1303, 0.1798, and 0.2281 values for nDCG@5, nDCG@10, and nDCG@20 respectively. This is lower than the baselines. The reason for the low performance of our models could be attributed to the lack of compute required to implement and train complex approaches.

Table 2: Ablation Results of Naive DDQN

Description	nDCG5	nDCG10	nDCG20
Max Exploit 5, Min Exploit 2	0.0784	0.1189	0.1895
Max Exploit 7, Min Exploit 5	0.0834	0.1296	0.2004
Max Exploit 10, Min Exploit 5	0.0908	0.1355	0.2041
Rewards without Zero Centering	0.0834	0.1296	0.2004
Zero Centered Rewards	0.0593	0.0966	0.2209
Binary Rewards	0.0745	0.1159	0.1831

Table 3: Ablation Results of Constrained Dynamic Action Space DDQN (CDAS-D)

Description (Embedding Size, Timesteps, Episodes)	nDCG5	nDCG10	nDCG20
Embeddings 64, 10 timesteps, 8400	0.1365	0.1799	0.2278
Embeddings 64, 10 timesteps, 14000	0.1303	0.1798	0.2281
Embeddings 64, 20 timesteps, 16800	0.1361	0.1806	0.2246
Embeddings 128, 20 timesteps, 22400	0.1210	0.1638	0.2142
Embeddings 256, 25 timesteps, 35000	0.1300	0.1755	0.2227

7 Conclusion and Proposed Extension

From our literature survey, we found that Reinforcement Learning methods are gaining traction in this domain due to their ability to deal with large amounts of states and actions. Thus we intended to test the hypothesis of Reinforcement Learning based Recommender Systems (RLRS) being able to bridge the dearth in traditional approaches using a Deep Q Network. When compared to baseline models, while the models don't achieve the expected performance of outperforming them, the initial results are promising with the loss reducing and nDCG increasing gradually. The agent might show significant improvements if trained for thousands of episodes as is the case with almost all reinforcement learning methods.

Currently only the IDs of movies and users have been embedded. Future work can explore the use of richer embeddings using the meta data provided in the dataset. Furthermore, different models such as LSTM, GRU etc can be used for embedding generation. Training the model for a large number of iterations could help as well. Overall, the work again establishes the importance of deep reinforcement-based recommender systems and their relevance to the current world.

References

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015. URL `https://arxiv.org/abs/1509.06461`.

David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35(12):61–70, dec 1992. ISSN 0001-0782. doi: 10.1145/138859.138867. URL `https://doi.org/10.1145/138859.138867`.

Paul Resnick and Hal R. Varian. Recommender systems. *Commun. ACM*, 40:56–58, 1997.

Michael J. Pazzani and Daniel Billsus. *Content-Based Recommendation Systems*, pages 325–341. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-72079-9. doi: 10.1007/978-3-540-72079-9_10. URL `https://doi.org/10.1007/978-3-540-72079-9_10`.

Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. *Content-based Recommender Systems: State of the Art and Trends*, pages 73–105. Springer US, Boston, MA, 2011. ISBN 978-0-387-85820-3. doi: 10.1007/978-0-387-85820-3_3. URL `https://doi.org/10.1007/978-0-387-85820-3_3`.

Francesco Ricci, Lior Rokach, and Bracha Shapira. *Introduction to Recommender Systems Handbook*, pages 1–35. Springer US, Boston, MA, 2011. ISBN 978-0-387-85820-3. doi: 10.1007/978-0-387-85820-3_1. URL `https://doi.org/10.1007/978-0-387-85820-3_1`.

J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132, 2013. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2013.03.012. URL `https://www.sciencedirect.com/science/article/pii/S0950705113001044`.

Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009, jan 2009. ISSN 1687-7470. doi: 10.1155/2009/421425. URL `https://doi.org/10.1155/2009/421425`.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Thorsten Joachims, Dayne Freitag, and Tom Mitchell. Webwatcher: A tour guide for the world wide web. *Proc 15th IJCAI*, 1, 10 1996.

Nima Taghipour, Ahmad Kardan, and Saeed Shiry Ghidary. Usage-based web recommendations: a reinforcement learning approach. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 113–120, 2007.

Tariq Mahmood and Francesco Ricci. Learning and adaptivity in interactive recommender systems. In *Proceedings of the ninth international conference on Electronic commerce*, pages 75–84, 2007.

Tariq Mahmood and Francesco Ricci. Improving recommender systems with adaptive conversational strategies. In *Proceedings of the 20th ACM conference on Hypertext and hypermedia*, pages 73–82, 2009.

Sungwoon Choi, Heonseok Ha, Uiwon Hwang, Chanju Kim, Jung-Woo Ha, and Sungroh Yoon. Reinforcement learning based recommender system using biclustering technique. *arXiv preprint arXiv:1801.05532*, 2018.

Jia-Wei Chang, Ching-Yi Chiou, Jia-Yi Liao, Ying-Kai Hung, Chien-Che Huang, Kuan-Cheng Lin, and Ying-Hung Pu. Music recommender using deep embedding-based features and behavior-based reinforcement learning. *Multimedia Tools and Applications*, 80(26):34037–34064, 2021.

Zhongqi Lu and Qiang Yang. Partially observable markov decision process for recommender systems. *arXiv preprint arXiv:1608.07793*, 2016.

Yang Zhang, Chenwei Zhang, and Xiaozhong Liu. Dynamic scholarly collaborator recommendation via competitive multi-agent reinforcement learning. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 331–335, 2017.

Xiangyu Zhao, Liang Zhang, Zhuoye Ding, Long Xia, Jiliang Tang, and Dawei Yin. Recommendations with negative feedback via pairwise deep reinforcement learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1040–1048, 2018.

Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. Drn: A deep reinforcement learning framework for news recommendation. In *Proceedings of the 2018 world wide web conference*, pages 167–176, 2018.

Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015.

Shi-Yong Chen, Yang Yu, Qing Da, Jun Tan, Hai-Kuan Huang, and Hai-Hong Tang. Stabilizing reinforcement learning in dynamic environment with application to online recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1187–1196, 2018.

F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), dec 2015. ISSN 2160-6455. doi: 10.1145/2827872. URL `https://doi.org/10.1145/2827872`.

Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering, 2017. URL `https://arxiv.org/abs/1708.05031`.

M Mehdi Afsar, Trafford Crump, and Behrouz Far. Reinforcement learning based recommender systems: A survey. *ACM Computing Surveys (CSUR)*, 2021.

Wang-Cheng Kang and Julian J. McAuley. Self-attentive sequential recommendation. *CoRR*, abs/1808.09781, 2018. URL `http://arxiv.org/abs/1808.09781`.

Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. *CoRR*, abs/2002.02126, 2020. URL `https://arxiv.org/abs/2002.02126`.

Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering github repo, 2018. URL `https://github.com/hexiangnan/neural_collaborative_filtering`.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013. URL `https://arxiv.org/abs/1312.5602`.

Appendix

Baseline Results

Since the authors have not released the best performing hyperparameters for Neural Collaborative Filtering Methods since, we trained several models to find the best performance. We were able to reproduce nDCG@10 results with a difference of 0.02. Although small, this difference in performance could be because we might not have found the optimal hyperparameters during our runs or due to random initialization of weights. Extensive hyperparameter tuning could reduce this error margin further. Our implementation of SASRec achieves a nDCG @ 10 score of 0.5629 which is very close to the reported score of 0.5905. Furthermore, the out of box implementation only computed the nDCG @ 10 metric. We modified this implementation to calcuate both nDCG @ 5 and nDCG @ 20. LightGCN achieved nDCG@5, nDCG@10 and nDCG@20 scores of 0.4358, 0.4046 and 0.3871 respectively. A thorough comparison of baseline results can be found at Table 1

Ablation Study

Apart from the hyperparameters, our ablation study also analyzes multiple configurations of the problem formulation. Here are some of the variants and the thought process behind their formulation.

- Handling exploration and exploitation:

  In order to force the model to explore all users equally, the step count dictionary was used as mentioned in the previous section. This has been noted as **max exploitation** meaning maximum number of timesteps of exploitation allowed on a user, and **min exploitation** which is the minimum number of timesteps that should be seen for each user.Although exploitation on one users state is present, samples were drawn from the experience replay buffer in random order.

- Using rewards as relevance score:

  Since the order of ranking (1st best recommendation to nth best recommendation) is done using Q values, there is a need to attach another quantity as the relevance of that recommendation from ground truth to evaluate the model using nDCG. In order to do this, the rewards were used as the relevance scores.

All the models have been trained for 5 epochs each except in the case of Max Exploit 10 and Min Exploit 5 where it was run for 3 episodes due to time constraints.

Challenges Faced

In the course of this experiments, we did encounter some challenges as listed below:

- While many applications deal with a static action space i.e we know the number of actions available to the agent, in this case the action space which are the movies is huge. To combat this we explored two routes - one for optimization and one for completeness as seen in the previous section.
- Slow run time due to sequential processing
- Multiple start states. Each users initial movie state can be different. This has been handled by storing the intial states of all the users in the experience replay during initialization.
- Ensuring all users are equally exploited on while drawing non correlated samples. This has also been resolved by using a dictionary to store the number of steps performed on each user and forcing a min and max exploitation rate during training.