Product Recommendation in Online Advertising with Reinforcement Learning

Carlos E. L. P. X. Torres

University of Colorado at Colorado Springs
1420 Austin Bluffs Pkwy.
Colorado Springs, Colorado 80918
clopespi@uccs.edu

Abstract

This work proposes a Recommender System focused on Product Recommendations in Online Advertisement. It presents a Deep Q-learning algorithm, with value learning, and Deep Q-Network methodology, to teach an agent by interacting with e-commerce websites. Then it processes those inputs and recommends products in another publisher website with the objective to maximize the click-through rate (CTR) of the ads displayed. A simulated environment, RecoGym, based on the Open AI gym framework, is used to illustrate the solution. The results achieved in the end by the agent show an average CTR of 2.13%, which is 159.75% better than the baseline random agent that has an average CTR of 0.82%.

1 Introduction

Online advertising started in 1994, when Wired.com, then known as HotWired, invented the web banner ad (Singel 2010). It made use of websites and other internet related content as an advertising medium, showing banners, or any other visual advertising peaces (text, display, affiliate links, videos), to the online content consumers.

Since its invention, online advertising had an explosive growth, diversifying its methods and approaches, and expanding to other internet connected devices different from the desktop, like smartphones, TVs, watches, and tablets. It has been also commonly referred to as internet advertising, online marketing, web advertising or online marketing.

With such growth and reach, online advertising started confronting significant problems. The advertisers, that pay the agencies that create and publish the ads, always want to maximize their marketing budget by showing ads to a very specific audience that is more likely to consume their products and services. Thus, the greatest challenge that the advertisement agencies have is to optimize the advertiser marketing budget by maximizing the click-through rate (CTR) on their ads. So the problem of product recommendation in online advertising started to gain more attention from computer scientists and, more specifically, machine learning specialists. With the goal to optimize and make advertising more intelligent, learning from user behavior patterns, how to understand and recommend the best ad, or product, to a specific audience or target group.

The problem of Product Recommendation in Online Advertising is part of a broader problem of optimal recommendation policies that Recommender Systems (Ricci, Rokach, and Shapira 2011) want to solve, through software tools and techniques providing suggestions for items to be of use to a user. They became more popular along the years in every aspect of the online life, applied to problems like what product to buy, what person to become friend, what news to read or what music to listen.

So the objective of this work is to propose a Recommender System focused on Product Recommendation for Online Advertising by presenting a Deep Q-learning algorithm, with value learning, and Deep Q-Network, methodology to teach an agent to process interactions via organic user sessions on an e-commerce website and maximize the click-through rate (CTR) in a publisher website for the recommendations of products for a user.

2 Background

In machine learning there are three main paradigms, supervised, unsupervised and reinforcement learning. Supervised learning needs a training set of labeled examples provided by a knowledgeable external supervisor. The labels are used to identify a category to which the situation belongs. Unsupervised learning is typically about finding structure hidden in collections of unlabeled data, but also needs a training set of data (Sutton and Barto 2018). The reinforcement learning paradigm is different from both because it does not necessarily require a training data set (labeled or for clustering) and can make an agent learn from interactions with an environment that the agent does not have any prior knowledge about. This approach is perfect for the problem of a recommender system focused on product recommendation in online advertisement, because it does not require previous knowledge of the environment, and the user interactions with the environment generate the input data necessary to train the agent. Although, in real-world applications, data sets of users' profiles and behaviors are necessary to train the agent, as observed later on section 4.10.

2.1 Reinforcement Learning

Reinforcement learning is a computational approach to learning from interaction through trial-and-error search and delayed reward (Sutton and Barto 2018). This is the approach that will be used on this work. The underlying mathematical model of reinforcement learning is the Markov deci-

sion process (MDP), defined by Szepesvári (2010) as a tuple $\{S, A, P, R, \gamma\}$, where S is the state space, A is the action space, P is the transition probability function, R is the reward function, and γ is the discount factor.

2.2 Problem Statement

The Recommender System problem here is formulated in terms of an MDP. It assumes the role of the agent, responsible for presenting online advertisements for the users in the system.

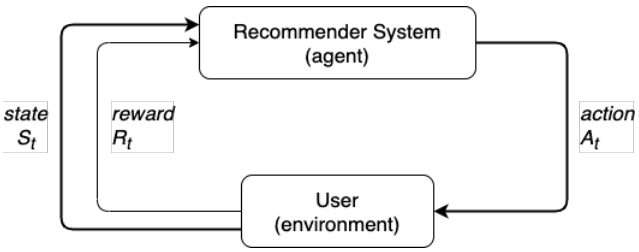

Figure 1: The Recommender System agent interaction in a Markov decision process

Figure 1 above illustrates the MDP for the problem. As He et al. (2020) did with their problem and Liu et al. (2020) presented in their preliminaries, below we explain the MDP components for this work's problem more in depth:

- *S* - The state space is represented by a finite set of users' profiles that are part of the environment with which the agent interacts. The agent will observe the users' behavior over a time period to learn from it.
- *A* - The action space is represented by a set of recommendations by the agent to the user on the publisher website, based on the views of the user of such item on the e-commerce website.
- *P* - The transition probability function defines how the state changes, given a certain action. It must satisfy the Markov property, ignoring the past and predicting the future based on the present. And it is represented as $P(s_t, s_{t+1}) = P(s_{t+1}|a_i, s_t)$ where $s_t \in S$ $a_i \in A$.
- *R* - The reward function represents the users' response to a recommendation from the agent, assuming a value of 1 if the user clicks on an advertisement presented by the agent, or 0 otherwise.
- γ - The discount factor or rate, can assume values $0 \leq \gamma \leq 1$. It increases the interest of the agent in having earlier rewards instead of future or later rewards. $\gamma = 0$ means the agent only focuses on immediate rewards, and $\gamma = 1$ means the agent prefers more future rewards.

The goal of a reinforcement learning agent is to find an optimal policy, a policy that maximizes the cumulative rewards at each state. In this context of online advertising, where we have a Recommender System as the agent, the goal is to maximize the cumulative reward for a given user, which means finding the optimal policy that maximizes the click-through rate (CTR).

3 Related work

Reinforcement learning for Recommender Systems is a well explored area. Advances in this field provide better recommendations for different online contents, like music, movies, videos, social media friends, news, and, in the context of this paper, products in online advertising.

Afsar, Crump, and Far (2021) observed that the recommendation problem was considered, traditionally, as a simple classification or prediction problem. Per contra, the sequential nature of the recommendation problem has been shown. Thus, it can be formulated as a Markov decision process (MDP) and reinforcement learning methods can be employed to solve it.

Zhao et al. (2021) and Zou et al. (2019) used deep reinforcement learning to create frameworks to optimize the long-term user engagement by continuously updating its advertising strategies. Both designed a Deep Q-Network architecture that takes charge of modeling complex user behaviors to find the optimal policy for the advertisement delivery.

A different approach to solve the problem of a recommender system as a Markov decision process (MDP) is taken by Choi et al. (2018). They propose a recommender system as gridworld game, using a biclustering technique, to address a problem they encounter of the large number of discrete actions. This approach aims to reduce the state and action space significantly.

Zheng et al. (2018) also used deep reinforcement learning but in the context of news recommendation. They proposed a Deep Q-learning recommendation framework to address the problem of boring news recommendation, that normally do not take in consideration user feedback (different from the click-through rate) and similar news recommended many times to the same user. They also use offline and online data sets from production environment of a commercial news recommendation application.

4 Methodology

In order to teach the agent on how to recommend products to users, we intend to use Q-learning. But, as our Recommender System agent will have to deal with variable and possible high volume data sets for the states and actions, instead of use a table to store and update the value function, we propose the use of a Deep Q-learning approach, using a Deep Q-Network to approximate the state-action value function.

4.1 Deep Q-learning

The Q-learning algorithm (Watkins and Dayan 1992) is a breakthrough in reinforcement learning, that came to drastically simplify the TD algorithms, directly approximating the q_* action-value function, independent of the policy being followed. It is defined as:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big] \tag{1}$$

where $max_a Q(S_{t+1}, a)$ is the max action-state value.

But, as previously stated, the agent will have to interact with an environment with thousands of states and actions

possibilities. This will prevent us from using a regular Q-learning algorithm, where a simple table is used to store and update the value function. Thus, causing problems of memory space and time to search the entire Q-table, becoming unfeasible. Instead, a neural network is used to approximate the Q-values.

4.2 Deep Q-Networks (DQN)

DQN were first used for playing Atari games (Mnih et al. 2013). They are artificial neural networks (ANN) used to approximate the state-action value function and update the parameter values, replacing the Q-table. An illustration of a DQN is shown on Figure 2.

The optimal state-action value function $Q^*(s, a)$ is the expected value of $R_{t+1} + \gamma max_a Q(S_{t+1}, a)$, from Equation 1, also known as the target Q-value for the network. The predicted Q-value is $Q(S_t, A_t)$.

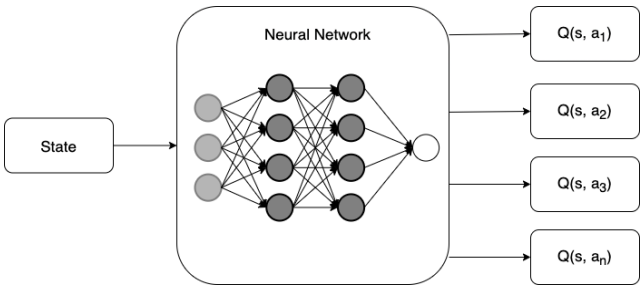

Figure 2: Deep Q-Network

4.3 Experience Replay

Similar to the way humans learn by using their memory of previous experiences, DQN can also make use of it. The Atari DQN work introduced a technique called Experience Replay to make the network updates more stable. This approach avoids computing the full expectation in the DQN loss by collecting the agent's experience $e_t = (S_t, A_t, R_t, S_{t+1})$, after every step t the agent performs and storing it in a replay memory $D_t = \{e_1, ..., e_t\}$.

Then, the replay memory D is used to train the DQN, by getting a random sample of experiences and using them as the input to the DQN.

4.4 Loss Function

The Q-network is a function approximator with weights θ. A Q-network can be trained by minimizing a sequence of loss functions $L_i(\theta_i)$ that changes at each iteration i. In this work, the loss function used initially was the Mean Squared Error (MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i' - Y_i)^2 \tag{2}$$

where Y_i' is the target value and Y_i is the predicted value.

During the learning process, we apply Q-learning updates, on samples of experience $(s, a, r, s^{'})$, randomly and uniformly taken from the replay memory $U(D)$. In our case, using the target and predicted values as the Q-values mentioned above, the loss function at each iteration i is shown below:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[(y_i - Q(S_{i-1}, A_{i-1}; \theta_{i-1}))^2\right] \tag{3}$$

where $y_i = R_i + \gamma max_a Q(S_i, a; \theta_i)$, and $i = t + 1$.

After several tests, a different loss function was considered when it started showing better results, the Huber loss function. It has been used in robust regression problems, and has very low sensitivity to outliers compared to the MSE, because it handles error as square only inside an interval. This function is quadratic for small values of $x = (Y_i' - Y_i)$, and linear for large values, as follows:

$$L_\delta(Y_i' - Y_i) = \begin{cases} \frac{1}{2}(Y_i' - Y_i)^2 & for |Y_i' - Y_i| \leq \delta \\ \delta|Y_i' - Y_i| - \frac{1}{2}\delta^2 & otherwise \end{cases} \tag{4}$$

where δ (delta) is a hyperparameter that can be set to adjust the interval in which the function will be quadratic or linear.

In other terms, for loss values less than δ (delta), it uses the MSE; for loss values greater than δ (delta), it uses the Mean Absolute Error (MAE). Thus, Huber loss provides the best of both MAE and MSE.

4.5 Algorithm

Algorithm 1 is used to implement the DQN, Deep Q-learning with Experience Replay (Mnih et al. 2013), and train our agent. Some modifications were made to adapt the algorithm to our problem.

The algorithm starts on line 1 by initializing the replay memory D and, on line 2, the state-value value function Q with random weights.

On line 3, starts the main loop of the algorithm, for all episodes until the maximum M. In each iteration, it initializes a sequence s_1 (Line 4) and starts the iterations through steps of each episode (Line 5), until the maximum T.

On lines 6 and 7, an action is selected using probability ϵ to choose a random action or the $max_a Q^*(\phi(S_t), a; \theta)$.

On line 8, the chosen action a is taken in the emulator, and it is obtained r_t and observation o_{t+1}.

On line 9, the next state s_{t+1} is set to s_t, a_t, o_{t+1} and ϕ_{t+1} is set to $\phi(s_{t+1})$.

On line 10, the transition is stored in the replay memory D. On lines 11 and 12, a random sample batch is obtained from D to be used to predict y_i and then train the target network with the Q-learning formula, if it is not a terminal state.

Finally, on line 13, the neural network gradient descent step is performed, calculating the loss.

4.6 Simulation

We are using a simulated environment called RecoGym (Rohde et al. 2018), an Open AI gym (Brockman et al. 2016), to train and test our agent. This environment allows, at evaluation time, the simulation of users' reaction to arbitrary recommendation policies, both organic and bandit user-item

Algorithm 1: Deep Q-learning with Experience Replay

Initialize the replay memory D to capacity N
Initialize the state-action value function Q with random weights
for all $episode = 1 \in M$ **do**
    Initialize the sequence $s_1 = o_1$ and the preprocessed sequence $\phi_1 = \phi(s_1)$
    for all $t = 1 \in T$ **do**
        With probability ϵ select a random action a_t,
        Otherwise select $a_t = max_a Q^*(\phi(S_t), a; \theta)$
        Execute the action a_t in the emulator and obtain the reward r_t and observation o_{t+1}
        Set $s_{t+1} = s_t, a_t, o_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
        Set $y_i = \begin{cases} r_j & \text{terminal } \phi_{j+1} \\ r_j + \gamma max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{non terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_i - Q(\phi_j, a_j; \theta))^2$
    end for
end for

interactions, and the relative impact of the user's past exposure level to ads on the click-through rate of an ad display at a given time.

In Figure 3, we show the RecoGym environment simplified model of user activity, alternating between organic e-commerce sessions and bandit publisher sessions, and finishing by stopping online activity for a longer period of time.

Figure 4 and Figure 5 show the simulation environment notation and example data, respectively.

The simulation starts with a user entering the organic session and observing different items at different time steps, within a variable time interval T. After that, the bandit session starts, where the recommender agent suggests several items to the user, based on their products' visiting history at previous time steps. The bandit session ends after a randomly chosen time period, different for each user. During the bandit session, the user may click on one of the recommended products. If this occurs, the recommender agent receives a reward and the user will move back to the organic session. Otherwise, the user is just moved back to the organic session. This process of transferring between the two sessions occurs for a random number of times for each user.

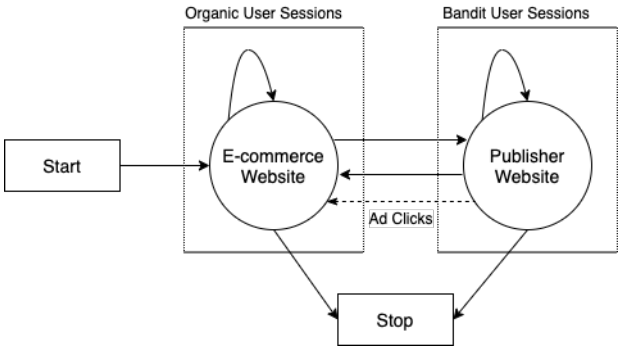

Figure 3: The RecoGym environment represented as a Markov chain of the organic and bandit user sessions. Adapted from RecoGym.

Symbol	Meaning
P	Number of products
u	User
t	Time index (discrete) starting from 0
$z_{u,t}$	Event t for user u is organic or bandit
$v_{u,t}$	Organic product view or undefined if $z_{u,t}$ = bandit
$v_{u,p,t}$	One hot coding of $v_{u,t}$ where p denotes the product
$a_{u,t}$	Recommended or action product or undefined if $z_{u,t}$ = organic
$c_{u,t}t$	Click was made on recommendation $a_{u,t}$ or undefined if $z_{u,t}$ = organic

Figure 4: RecoGym notation. Adapted from RecoGym.

u	t	$z_{u,t}$	$v_{u,t}$	$a_{u,t}$	$c_{u,t}$
.	.	.	.	.	.
10	0	organic	104	NA	NA
10	1	organic	52	NA	NA
10	2	organic	71	NA	NA
10	3	bandit	NA	42	0
10	4	bandit	NA	52	1
10	5	organic	52	NA	NA
.	.	.	.	.	.

Figure 5: RecoGym example data. Adapted from RecoGym.

4.7 Training Process

To train our recommender agent model, we deploy the DQN using an artificial neural network (ANN) to approximate the state-action value function. Table 1 shows the hyperparameters of the DQN that are used to obtain the current results. There parameters can be adjusted for future improvements of the model training.

Hyperparameter	Value
Alpha (learning rate)	0.001
Gamma (discount factor)	0.95
Delta (Huber loss)	1.5
Epsilon (exploration rate)	0.8
Epsilon min	0.001
Epsilon decay	0.995
Replay Memory	1,000,000
Batch size	64
Training users	1,000
Number of products	10
Episodes	200

Table 1: Hyperparameters of the DQN.

The state is preprocessed each time before sending to the DQN to simplify the input. A tuple $(u, v_{u,t})$, representing the User and the Product viewed by the user, is created and used as the state. The action space is the number of products used in the simulation, and each action is a product_id, representing the recommendation of a product for the user. Then, a reward = 1 is generated in case of a user click on the recommended product, otherwise a reward = 0 is generated.

4.8 System Architecture

The complete system architecture of our recommender agent is presented in Figure 6. The internal DQN architecture is shown in Figure 7.

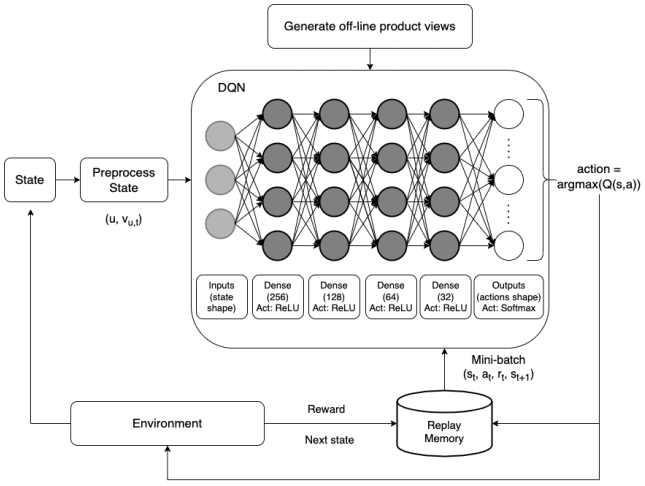

Figure 6: System architecture.

Before the training starts, the system generates an initial product views off-line list from 1000 users. This is created using the simulated environment function to create product views for users in organic sessions. We use those product views to feed the DQN to pull random products (actions) from it using better informed probabilities, based on the number of product views for each product, not just totally random actions.

The states are preprocessed before being fed to the DQN, to clean up unused data and simplify the input shape, as described on Section 4.7.

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 256)	768
dense_1 (Dense)	(None, 128)	32896
dense_2 (Dense)	(None, 64)	8256
dense_3 (Dense)	(None, 32)	2080
dense_4 (Dense)	(None, 10)	330
Total params:	44,330	
Trainable params:	44,330	
Non-trainable params:	0	

Table 2: DQN model summary.

The DQN is a Sequential model summarized in Table 2. It has an Input layer with the shape of the state (2); 4 Dense layers with 256, 128, 64, and 32 nodes with *ReLU* activation function; and an Output layer with the shape of the action space (number of products) with a *Softmax* activation function. This model configuration was selected according to the shape of the inputs, that are one-dimensional arrays. Other networks, like a convolutional neural network (CNN), were considered during the research. But a CNN, for example, is more useful when the input shape is grid-like, two-dimensional matrices like images.

The Replay Memory is updated as described in Section 4.3 and provides mini-batches of random samples to the DQN.

Internally, the DQN uses a Prediction network, that updates the weights after N steps, and a Target network that outputs the actual Q-value. With the outputs of both, the Target Q-value and Predicted Q-value, the loss is calculated and the DQN can learn.

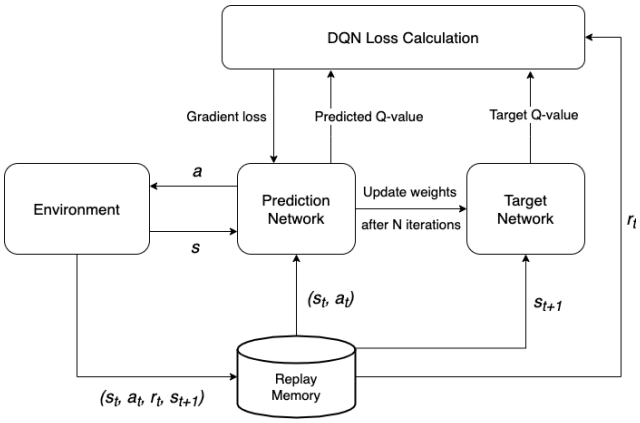

Figure 7: DQN internal architecture. Adapted from Nair et al. (2015).

4.9 Implementation

The training algorithm is implemented in Python 3 and uses the Tensorflow Keras library for the deep learning capabil-

ities. We also use Numpy for numeric functions and Recogym Environment and Agent libraries for the simulation.

4.10 Data

There is no labeled or clustered training data for the agent, once the Recommender System will be under a model-free algorithm, where it has no knowledge of the past states, or how the environment works, or the next observation without taking an action and getting a feedback, a new state and possibly a reward, from the environment. But there is data generated by the environment (observations), that represent the user interactions with the websites, either organic product views on the e-commerce website or clicks on the publisher website. This data is generated by the RecoGym gym during the state observations. In real-world applications, this data is acquired by harvesting users' navigation through websites, using cookies and track scripts. This data is not easy to obtain, because companies that produce and process it charge a lot for a good data set with users' profiles and behaviors.

5 Evaluation

In order to evaluate how the agent performed in learning how to recommend products to the user, we track the click-through rate (CTR) during the policy learning, until it stops improving, then we know we have the best policy at the end of the training. According to GoogleAds (2021), a good CTR is generally considered to be 1% or higher. A chart of the click-through rate is also be presented to easily visualize the learning process. Below is how the CTR is calculated.

$$\text{CTR} = \frac{\text{Total Measured Ad Clicks}}{\text{Total Measured Ad Impressions}} \times 100 \quad (5)$$

6 Results and Discussion

Our recommender agent training results are presented in Figure 8. The results for a random agent are shown in Figure 9 for comparison. The figures show the CTR calculated after each episode (blue) and the average CTR obtained after 200 episodes (orange) for each agent.

The recommender agent results were obtained using the Huber loss function from Equation 4, the system architecture described in Figures 6 and 7, and the hyperparameters showed on Table 1. These results were significantly better than the previous Midterm ones, with an average CTR = 1.69%. There, the MSE loss function and a different system architecture for the neural network, with fewer layers, were used. Adjustments in the hyperparameters were also made to contribute with the better results.

7 Future

Besides continuing to improve the DQN architecture and the hyperparameters to obtain better results, we would like to add a supervised learning component to the solution. This would create a hybrid system, using real-world data, as mentioned in section 4.10, to train a preliminary model that would be passed as a starting point to the DQN. This approach can bring significant benefits to the reinforcement learning agent, cutting training time and having even better results with fewer episodes.

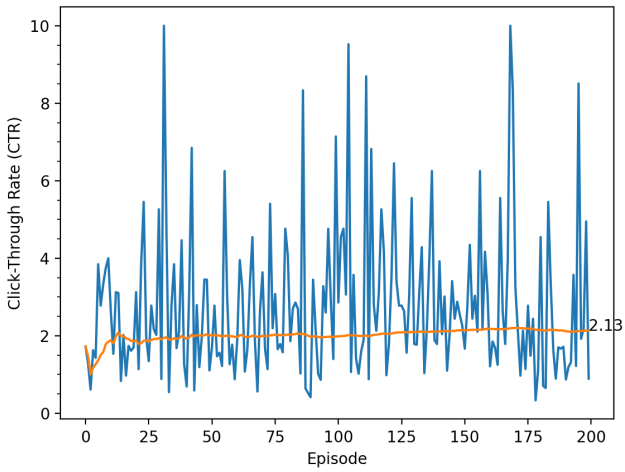

Figure 8: DQN agent results. Average CTR = 2.13%

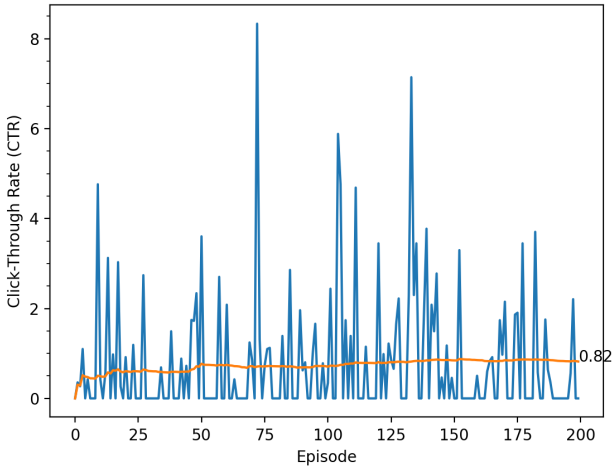

Figure 9: Random agent results. Average CTR = 0.82%

8 Conclusion

In this paper, it is proposed the creation of a reinforcement learning agent as a Recommender System to learn by interacting with user inputs in an e-commerce website to recommend items in another publisher website through online advertising with the objective to maximize the click-through rate (CTR). A Deep Q-learning algorithm, with value learning, and Deep Q-Network, is used to teach the agent the best policy. A simulated environment is used to illustrate the results, RecoGym, based on the Open AI gym framework. The results achieved in the end by the agent show an average CTR of 2.13%, which is 159.75% better than the baseline random agent that has an average CTR of 0.82%.

References

Afsar, M. M.; Crump, T.; and Far, B. H. 2021. Reinforcement learning based recommender systems: A survey. *CoRR*, abs/2101.06286.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI Gym. *CoRR*, abs/1606.01540.

Choi, S.; Ha, H.; Hwang, U.; Kim, C.; Ha, J.; and Yoon, S. 2018. Reinforcement Learning based Recommender System using Biclustering Technique. *CoRR*, abs/1801.05532.

GoogleAds. 2021. Measure traffic to your website. https://support.google.com/google-ads/answer/1722035. Accessed: 2021-10-20.

He, Z.; Tran, K.-p.; Thomassey, S.; Zeng, X.; and Yi, C. 2020. A reinforcement learning based decision support system in textile manufacturing process. In *Developments of Artificial Intelligence Technologies in Computation and Robotics: Proceedings of the 14th International FLINS Conference (FLINS 2020)*, 550–557. World Scientific.

Liu, F.; Tang, R.; Li, X.; Zhang, W.; Ye, Y.; Chen, H.; Guo, H.; Zhang, Y.; and He, X. 2020. State representation modeling for deep reinforcement learning based recommendation. *Knowledge-Based Systems*, 205: 106170.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. A. 2013. Playing Atari with Deep Reinforcement Learning. *CoRR*, abs/1312.5602.

Nair, A.; Srinivasan, P.; Blackwell, S.; Alcicek, C.; Fearon, R.; Maria, A. D.; Panneershelvam, V.; Suleyman, M.; Beattie, C.; Petersen, S.; Legg, S.; Mnih, V.; Kavukcuoglu, K.; and Silver, D. 2015. Massively Parallel Methods for Deep Reinforcement Learning. *CoRR*, abs/1507.04296.

Ricci, F.; Rokach, L.; and Shapira, B. 2011. Introduction to recommender systems handbook. In *Recommender systems handbook*, 1–35. Springer.

Rohde, D.; Bonner, S.; Dunlop, T.; Vasile, F.; and Karatzoglou, A. 2018. RecoGym: A Reinforcement Learning Environment for the problem of Product Recommendation in Online Advertising. *CoRR*, abs/1808.00720.

Singel, R. 2010. OCT. 27, 1994: Web gives birth to banner ads. *Wired. com*.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Szepesvári, C. 2010. *Algorithms for Reinforcement Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Watkins, C. J.; and Dayan, P. 1992. Q-learning. *Machine learning*, 8(3-4): 279–292.

Zhao, X.; Gu, C.; Zhang, H.; Yang, X.; Liu, X.; Liu, H.; and Tang, J. 2021. DEAR: Deep Reinforcement Learning for Online Advertising Impression in Recommender Systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 750–758.

Zheng, G.; Zhang, F.; Zheng, Z.; Xiang, Y.; Yuan, N. J.; Xie, X.; and Li, Z. 2018. DRN: A deep reinforcement learning framework for news recommendation. In *Proceedings of the 2018 World Wide Web Conference*, 167–176.

Zou, L.; Xia, L.; Ding, Z.; Song, J.; Liu, W.; and Yin, D. 2019. Reinforcement learning to optimize long-term user engagement in recommender systems. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2810–2818.