Empirical Study of a Deep Reinforcement Learning Method for Restaurant Recommendation Systems

Vele Tosevski
University of Toronto
vele.tosevski@mail.utoronto.ca

Keegan Poon
University of Toronto
keegan.poon@mail.utoronto.ca

Abstract

As cities grow larger, residents are flooded with more restaurant options, increasing indecisiveness in choosing a restaurant to visit or order from. Various models for separating and recommending general items have been developed and are used [1], but the most common types are limited in various ways. They assume the recommendation process is static as opposed to dynamic between the user and the recommendation system and are tunnel-visioned on the next recommendation without considering future consequences. We examine the DRR framework [2] as it applies to restaurant recommendations. Using the Yelp dataset [3], we achieve a best average reward of 0.65 and best precision of 0.60.

1 Introduction

Restaurant recommendation is becoming more important as cities grow and the amount of restaurant possibilities grow. Companies such as Uber Eats and Doordash already have recommendation systems in place to help users decide which restaurants to order from. Companies such as Tripadvisor also provide recommendations for restaurants to visit. Most of the companies we studied use supervised and unsupervised machine learning (ML) models to provide recommendations while none use reinforcement learning (RL) models (according to the data made available). Recommendation seems to fit in a natural RL setting where users and a recommendation model maintain a symbiotic relationship: the model provides recommendations to users, helping them come to a decision, and users provide feedback to the model, helping the model make better recommendations. As such, we decided to research RL methods for recommendation and select one to evaluate its effect on restaurant recommendation.

One of the leading frameworks we researched is the **D**eep **R**einforcement Learning based **R**ecommendation Framework with Explicit User-Item Interactions Modeling (DRR) [2]. According to the paper, the framework outperforms current (as of 2019) recommendation systems on four datasets. As a result, we decided to test this framework's performance on restaurant recommendation (effectively, a fifth dataset).

In this paper, we investigate the application of DRR on restaurant recommendation. The following sections detail the formulation of the restaurant recommendation problem as an MDP, a summary of the DRR algorithm, our methods, and evaluations.

2 Problem

We wish to create a dynamic recommendation system for restaurants given a user's personal history and preferences. This has applications in delivery or ordering applications such as Uber Eats and restaurant recommendation services such as Tripadvisor, potentially boosting usage and sales given good performance. We choose to model this as an RL problem due to the lack of capability in existing

systems to model user and item interactions in the learning process, hoping that this addition would perform better than the state of the art.

2.1 Specification

We model this problem as an MDP where:

1. the **agent** is the recommendation system,
2. the **environment** is a user visiting restaurants,
3. the **state** is a representation of a user's demographics and most recent positively rated items,
4. **actions** are a continuous vector weighting restaurants in a candidate set,
5. the **reward** is a user's rating of a visited restaurant,
6. **transitions** are determined once a user provides feedback,
7. and γ is a discount factor determining the importance of future ratings.

Figure 2 in [2] shows the MDP process pictorially. Using the above definitions, restaurant recommendation follows the same process which is why this framework applies theoretically. Our evaluation results verify this empirically.

3 Prior Work

3.1 Non-RL Recommendation Systems

There are a whole plethora of techniques used outside of RL such as content-based filtering [4], matrix factorization [5], logistic regression [6], factorization machines [7] and deep learning models as well.

Some of the earlier work on this subject used content filtering or collaborative filtering [4] to generate recommendations with people that have similar taste, but for problems with large amounts of items, the sparsity of the data poses a challenge as there are not as many users sharing a large overlap of items which does not allow us find similarities very easily and when we do, it's very unreliable since we have so few data points. To solve this matrix factorization techniques [5] have been applied in collaborative filtering methods help deal with the lack of data, throwing the user and item data into one vector space and then factorizing this into multiple matrices of inferred relations between the users and items. Another attempt is using factorization machines, or the extended field-aware factorization machines[7] or even deep learning models to model feature interactions by themselves.

There have also been attempts that define it as a binary classification problem, in which various regression models can be used [6], but it suffers from overfitting in training, where it is difficult for the model to adapt to new feature interactions not seen before.

Aside from their individual issues, they all have common problems of not being dynamic systems able to capture how updates and future events may factor into the current model [2]. For example, if the users preferences changes over time then the models are going to break much more easily. This is why further methods are used in solving this problem.

3.2 RL Recommendation Systems

The general recommendation problem has been modeled with RL techniques as well. Using policy based approaches [8], these generally use deterministic policies to generate actions, however this comes at a cost for large action spaces. In this case they turn to modelling the recommendation items in a continuous space and have the policies pick the closest one using a neighbourhood method, but the continuous-discrete gap may throw off performance. One other point about the policy based approaches is also that none of them pay attention to the state representations.

In value based approaches [9], they follow the Q-value for all actions going to the argmax of Q. Various methods are used to model Q values such as Dueling Q-network in [9] with some other proposed updates such as dueling bandit gradient descent. but trying to evaluate Q over every action when we have immense possibilities for these is not very efficient.

3.3 DRR Systems

In DRR [2], the authors propose an actor-critic algorithm similar to DDPG [10]. In this case they use an actor-critic model with deep neural nets to estimate the policy and value function as well as a state representation model to let the model learn as well as possible from feature interaction as well. They use this algorithm to train the framework with their additional state model added to the actor and critic.

The feature of the state representation model allows them to show how the elements of the dataset are correlated with each other in various ways making learning more efficient. [11] In the paper they provide multiple methods of doing so, element wise multiplication of each item pair, concatenating user and items, and the best performing DRR-ave which performs average pooling on the previous items, multiplies it with the user and concatenates the user, averaged items and multiplied user and items all together as the new state.

This is as opposed to DRR-p or DRR-u which showcase only user-item feature interactions or feature interactions between items respectively. In their benchmark, save for one example, DRR-ave beat out the the competition as well as the other state representation models showing the importance of a good state representation.

We choose to re-implement this algorithm to test it's performance on a different dataset since in their case they are looking at purely digital environments such as music or movies. This means that the worst case for a bad recommendation is the user dislikes the item, but in our case we would have no guarantee that our user would be able to visit the restaurant recommended whether it be to distance or other hard factors.

4 Empirical Testing

4.1 Dataset

We used the Yelp dataset [3], which consists of Users, Businesses and Reviews with other customer related features which we did not use in our experiments. To reproduce the work of the DRR paper as close as possible, we only took the user, review and business identifiers of all restaurant reviews as well as their timestamp to order in the history buffer. To do this, the business data was filtered by the Restaurant attribute before joining with reviews and users dropping that which isn't relevant to the specified businesses. The rest of the features were generated with these identifiers using a PMF [12] before being used in our algorithm.

The original review ratings range between [1, 5] as discrete integer values. To stabilize training, we normalize the review ratings to values within [-1,1]. This allows us to easily identify negative from positive ratings. We normalize using the following formula, also used in [2]:

$$R(s, a) = \frac{1}{2}(rating_i - 3)$$

where $rating_i$ is the rating of the item recommended after being in state *s* and taking action a.

Users	Businesses	Ratings
2,189,457	160,584	8,635,403

4.2 Method

4.2.1 State representation

As mentioned previously, we are keeping close to the original DRR paper and choose to implement DRR-ave for our state representation. The method described concatenates three vectors: (1) user embedding, (2) average pooling of the embeddings of the *n* most recently predicted, positive items, (3) the first vector multiplied elementwise with the second. Figure 6 in [2] shows a diagram of DRR-ave worth observing.

We choose to ignore the DRR-u and DRR-p state representations, since their performance was much less in comparison, according to the authors' evaluation results. DRR-ave allows us to compare the

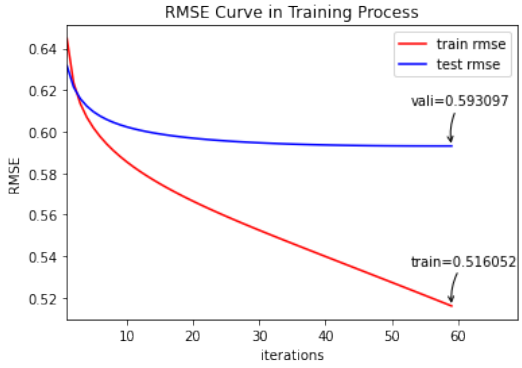


Figure 1: RMSE Training and Validation Loss of PMF

highest performing version of the algorithm in our case, and from the formation of the other state representations, it seems unlikely that they would provide any benefit, as DRR-ave combines their advantages of linking users to items, and representing interactions between items.

4.2.2 Prioritized Experience Replay

As in the paper, we utilize a Prioritized Experience Replay buffer [13] which replays the transitions for which the TD error is maximized. Using this heuristic we can prioritize learning from the "strangest" cases from the perspective of our action-value function. Given a priority p_i of a specific example, the chance that we sample it is

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

with $\alpha > 0$ being a hyper parameter chosen for our replay buffer. This way, the probability is monotonic with respect to its priority ensuring that we get the higher TD error elements more at a rate controlled by α. If α was 0, this would simply revert to the uniform case without any items getting priority.

4.2.3 Probabilistic Matrix Factorization (PMF)

Due to the sparse nature of user reviews for a large swath of businesses, PMF [12] is to both predict the review scores where there are none given, as well as provide the embeddings for the users and items which are used in training. PMF embeddings are learned representations of users and items using only a couple of real features in the dataset. The only features we used to learn the embeddings are the index identification numbers of the users and items and ratings of each item by each user.

For training the PMF, we used a batch size of 100,000, training, validation, test ratio of 0.8, 0.1, and 0.1 respectfully, and stochastic gradient descent with 0.1 weight decay, a 1e-4 learning rate, and 0.9 momentum. The training lasted 56 epochs. The model was from [12] with two added bias vectors to the end of the forward pass, one for the user, one for the item. The embedding size was set to 100, the same as the best embedding size found in [2]. Figure 1 shows training and validation loss as RMSE loss, or root mean squared error loss. The model performs satisfactorily with a test loss of 0.59.

It is important to note that the authors do not discuss the PMF model they used. We had to derive it on our own and conduct testing. Because it is not the model we are evaluating in this report, we did not do extensive testing and relied on a couple test runs. We realize that since we use the PMF in the DRR training, the performance of the DRR depends on the performance of the PMF. We leave further testing and defining a more accurate model to future work.

4.2.4 Actor and Critic Networks

Figure 3 in [2] shows a diagram of the model architecture. To summarize, the actor model receives a state outputted by the state representation module (DRR-ave in this case) as its input. It feeds the state through various hidden layers and outputs a continuous action vector (same shape as item embedding

vector). To recommend an item, the algorithm matrix multiplies a candidate set of item embeddings by the action vector and recommends the argmax of the resulting vector.

The critic network takes the actor's action and state representation vector as an input. It feeds the concatenated vector through a series of hidden layers and outputs a scalar ($Q(x, a)$).

4.2.5 The Algorithm

The algorithm is based on the DDPG [10] algorithm. It updates the actor network with a stochastic policy gradient, the state representation network using the actor's signal, and the critic network using the L2 loss between the target critic network and the current critic network. It updates the target networks using a soft-update (weighted by τ) at every learning step. It samples using Prioritized Experience Replay to perform learning updates and uses a pre-trained PMF as the reward signal when ratings are not available (uses the normalized rating otherwise). Algorithm 1 in [2] shows the full algorithm.

4.3 Evaluation

In our evaluations we train with the following hyper parameters

- $batchsize = 64$
- $\tau = 0.01$
- $\alpha = 0.3$
- $\gamma = 0.9$
- $\beta = 0.4$

We train with two Adam optimizers, one for the actor network and another for the critic network. The learning rate for the actor network is 1e-4 and 1e-3 for the critic network. We utilize L2 regularization with a weight decay of 0.01 for both. The size of the history buffer is empirically set to 5 as in the paper. The episode length, or the number of sequential recommendations for a user, is also set to 10 as in the paper. In each recommendation episode, we do not recommend items we have already recommended. We start learning after 5000 timesteps (recommendations) to build the replay buffer. The replay buffer is set to a maximum size of 100,000. We train for 260,000 timesteps and use a linear, decreasing ϵ value for ϵ-greedy action choice, starting at 1.0, and decreasing to 0.1 over 10,000 timesteps. The reward signal is the normalized rating for the user and item pair if the user rates the recommended item, otherwise, it is the result of a forward pass of the pre-trained PMF with the user index and recommended item index as an input.

We train on a randomly sampled 80% of the dataset, reserving the rest for evaluation. Both offline and online performance is evaluated, offline using Average Precision @K and online using Average Reward @K metrics to compare performances. For an offline evaluation, Average Precision @K is the amount of positive predictions made divided by the total amount of predictions made (*K*) for a user, for 20 runs of evaluation over 500 users. Average Reward @K is the average reward per user receiving K recommendations for a total of 20,000 recommendations over many users, repeated 20 times. We use these metrics since they are used by the paper to evaluate the model over the datasets the authors used, providing a good performance comparison. We do not evaluate the NDCG@K metric also evaluated in the paper due to time constrictions and leave it for future work. We calculate the means of the 20 runs for each *K* and standard deviations to represent the error bounds of the means.

The offline and online evaluation procedures follow the procedures outlined in [2] precisely. To summarize, for the offline evaluation, it follows Algorithm 2 in [2] where recommendations are made and judged from the set of items the user has rated rather than all candidate items. The online evaluation follows the training algorithm with the only difference being the networks and their parameters are reset to those of the learned networks before each episode (user recommendation period). This is close to a real scenario where users start with the base model and continue training or refining the model with their own interactions.

We perform offline evaluations of the PMF, to serve as a foreign method benchmark, and the learned model. We perform an online evaluation of the learned model. We compare the metric results with those in the paper. We train two different DRR models with different seeds (0 and 1) to analyze the stability of our model.

Table 1: Offline Evaluation Results

(a) Offline PMF Average Precision @K

K	Average Precision	σ
5	**0.7283**	0.0097
10	0.5597	0.0062
15	0.4587	0.0070
20	0.3999	0.0060

(b) Offline DRR Average Precision @K (Seed 0)

K	Average Precision	σ
5	**0.6005**	0.0119
10	0.4752	0.0070
15	0.3987	0.0069
20	0.3555	0.0064

(c) Offline DRR Average Precision @K (Seed 1)

K	Average Precision	σ
5	**0.6013**	0.0156
10	0.4748	0.0053
15	0.3975	0.0062
20	0.3472	0.0046

Table 2: Online Evaluation Results

(a) Online DRR Average Reward @K (Seed 0)

K	Average Reward	σ
5	0.6151	0.0042
10	**0.6544**	0.0082
15	0.6479	0.0080
20	0.6348	0.0089

(b) Online DRR Average Reward @K (Seed 1)

K	Average Reward	σ
5	0.5546	0.0039
10	0.5924	0.0076
15	0.6169	0.0076
20	**0.6208**	0.0072

It takes around 1 hour and 30 minutes for training, 1 hour and 30 minutes for each offline evaluation, and 6 hours and 30 minutes for online evaluation. While we trained around 30 different models with different hyperparameters, finally reaching the best performing ones, we only evaluated two models because it takes about 11 hours to complete training, two offline evaluations, and one online evaluation together using one GPU.

We trained and evaluated both the PMF and DRR on one Google Colab Pro GPU (Tesla V100-SXM2-16GB). The source files are located in the following repository: https://github.com/irskid5/drr_restaurants.

4.3.1 Training

Figures 5 and 6 show that training converges for both the actor and the critic networks (DRR-ave is included in the actor network since the actor signal is used to update the DRR-ave network). We also see that loss is almost identical in both seeds, suggesting that the algorithm converges well regardless of seed. The plots are smoothed with the outside noise being the actual loss and the inside line being a mean convolution kernel applied to the real loss.

Figure 2: PMF Evaluation Graph

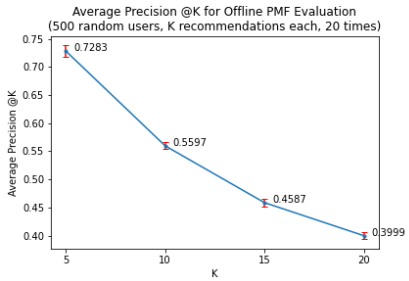
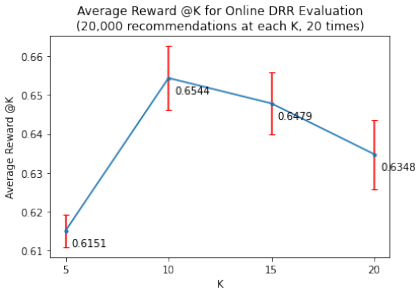
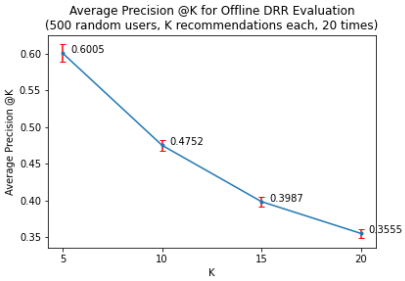


Figure 3: DRR Evaluation Graphs (Seed 0)

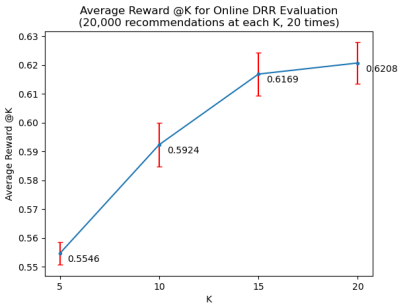


(a) DRR Online Evaluation Graph

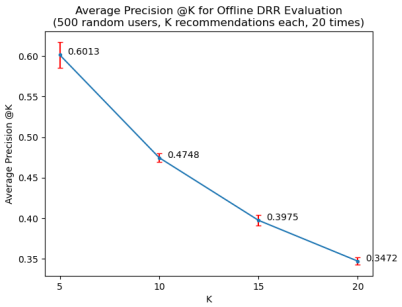


(b) DRR Offline Evaluation Graph

Figure 4: DRR Evaluation Graphs (Seed 1)

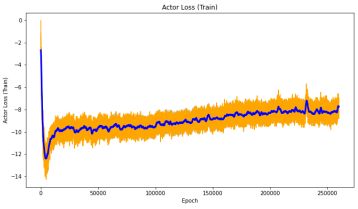


(a) DRR Online Evaluation Graph

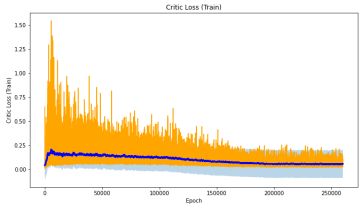


(b) DRR Offline Evaluation Graph

Figure 5: Actor and Critic Loss Seed 0

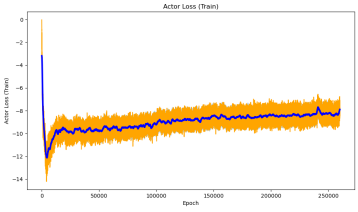


(a) Actor Loss

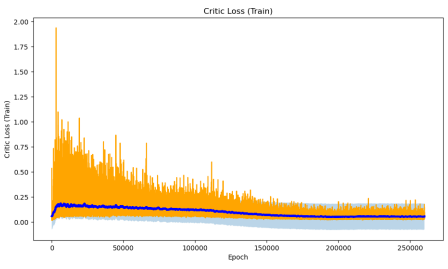


(b) Critic Loss

Figure 6: Actor and Critic Loss Seed 1



(a) Actor Loss



(b) Critic Loss

4.3.2 Baseline PMF

The baseline performance for offline recommendations seems to work best with a small K, immediately dropping off in performance for higher values. This behaviour is not too surprising and we see similar results for the ML 100k and ML 1m data sets in the original paper [2]. This suggests that models like PMFs maximize performance for immediate returns rather than future returns. Although it performs the best with respect to Precision@K, it does not have an application in the online setting and is therefore not suggested for restaurant recommendation. It serves only as a good benchmark for the offline evaluation of DRR.

4.3.3 Offline Evaluation

The offline performance seems to be extremely consistent in both Figures 3b and 4b with very low variation, remaining almost exactly the same across both seeds. The standard deviations are low suggesting a good performance accuracy based on K. Examining our offline evaluation, we seem to be generating a small number of good recommendations in a row and then as k increases, our performance decreases. This also seems consistent with our baseline and the results in the paper, if not falling slightly short of them. It also seems that performance starts to level off around $K = 20$ suggesting that higher values would achieve around the same performance. To reiterate, the paper does not discuss its evaluation results with respect to its PMF evaluation. It is conceivable that a better PMF model would make our DRR model perform better with respect to precision.

Although decreasing precision is expected with a larger recommendation pool, the difference between the paper results and these results is concerning. It seems as though there is a bias of -0.1 added to our results. That being said, the precision still seems relatively reasonable in how it changes with respect to K, compared to other datasets.

4.3.4 Online Evaluation

As we can see in figure 4a, we get a contrasting result for online evaluation between both seeds. This seems to suggest that since a random number of users is selected at each evaluation, some noise is introduced between different seed evaluations. For seed 0, we achieve a strong average reward at $K = 10$ while for seed 1, we achieve a strong average reward at $K = 20$. Both are, however, within 0.03 of each other, suggesting they perform similarly. We choose to disregard this slight difference given how all the other evaluations produce almost identical results across different seeds.

The results are very promising. The minimum reward is -1 while the maximum is 1. Both of our seeds hover around 0.61 and 0.62 which suggests the model performs well on the average in an online setting. The standard deviations are small as well suggesting the model is accurate in its successive predictions. It is difficult to compare the online results to the ones in the paper since the paper does not provide average reward, rather, it provides total reward. That being said, the online evaluation offers a better picture of the performance of DRR on restaurant recommendation than the offline evaluation. Since the ideal scenario is to apply this in an online setting, the fact that the online performance seems to be positive in general gives a good outlook on the algorithm's application to restaurant recommendation.

5 Conclusion

In this work, we model restaurant recommendation as an MDP to which we can apply a reinforcement learning algorithm to learn interactions between users and restaurants. We chose to apply the DRR model and evaluate its effect in both online and offline settings. The model does not perform as well as we expected in the offline setting. The online setting, which is the real setting a restaurant recommendation algorithm would be set in, provides a much better result suggesting that DRR with DRR-ave performs well for the task of restaurant recommendation.

In the past year (2020), the authors have released an updated paper [14] which uses an attention layer (DRR-att) for the state representation and more metrics for evaluation. In the future, we plan on implementing DRR-att and evaluating the model with the metrics used in [14]. We also plan on investigating how we can use more data from the dataset to create embeddings rather than only the ratings and user/item IDs. It seems rather wasteful to not use all the data collected in the dataset.

References

[1] R. Mu, "A survey of recommender systems based on deep learning," *IEEE Access*, vol. 6, pp. 69 009–69 022, 2018.

[2] F. Liu, R. Tang, X. Li, W. Zhang, Y. Ye, H. Chen, H. Guo, and Y. Zhang, "Deep reinforcement learning based recommendation with explicit user-item interactions modeling," 2019.

[3] Y. Inc., "Yelp open dataset." [Online]. Available: https://www.yelp.com/dataset

[4] Y. Deldjoo, M. Elahi, P. Cremonesi, F. Garzotto, P. Piazzolla, and M. Quadrana, "Content-based video recommendation system based on stylistic visual features," *Journal on Data Semantics*, vol. 5, no. 2, pp. 99–113, Jun 2016. [Online]. Available: https://doi.org/10.1007/s13740-016-0060-9

[5] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[6] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, S. Chikkerur, D. Liu, M. Wattenberg, A. M. Hrafnkelsson, T. Boulos, and J. Kubica, "Ad click prediction: a view from the trenches," *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013.

[7] Y. Juan, Y. Zhuang, W.-S. Chin, and C.-J. Lin, "Field-aware factorization machines for ctr prediction," 09 2016, pp. 43–50.

[8] Y. Hu, Q. Da, A. Zeng, Y. Yu, and Y. Xu, "Reinforcement learning to rank in e-commerce search engine: Formalization, analysis, and application," 2018.

[9] G. Zheng, F. Zhang, Z. Zheng, Y. Xiang, N. J. Yuan, X. Xie, and Z. Li, "Drn: A deep reinforcement learning framework for news recommendation," in *Proceedings of the 2018 World Wide Web Conference*, ser. WWW '18. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2018, p. 167–176. [Online]. Available: https://doi.org/10.1145/3178876.3185994

[10] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2019.

[11] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He, "Deepfm: A factorization-machine based neural network for ctr prediction," in *IJCAI*, 2017.

[12] A. Mnih and R. R. Salakhutdinov, "Probabilistic matrix factorization," in *Advances in Neural Information Processing Systems*, J. Platt, D. Koller, Y. Singer, and S. Roweis, Eds., vol. 20. Curran Associates, Inc., 2008. [Online]. Available: https://proceedings.neurips.cc/paper/2007/file/d7322ed717dedf1eb4e6e52a37ea7bcd-Paper.pdf

[13] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2016.

[14] F. Liu, H. Guo, X. Li, R. Tang, Y. Ye, and X. He, "End-to-end deep reinforcement learning based recommendation with supervised embedding," in *Proceedings of the 13th International Conference on Web Search and Data Mining*, ser. WSDM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 384–392. [Online]. Available: https://doi.org/10.1145/3336191.3371858