CS5446 Project Report: Deep Reinforcement Learning based Recommendation for Restaurant Self Order System

Team P23

Le Tan Dang Khoa	E1124464	A0274718U
Zheng Zhiqing	E0293907	A0091663B
Thong Chee Wai, Benjamin	E1101843	A0268544U
Chuang Fengchia	E1327947	A0290916X

Abstract

Table turnover rate is a key performance indicator for the Food and Beverage (F&B) industry. To help improve the metric, this paper details a deep reinforcement learning based recommendation system for self ordering that was trained on 11 months of actual customer order data. The reinforcement learning agent is designed to learn and adapt different user profiles as well as their ever-changing preferences by utilizing the Critic-Actor framework. The efficacy of the reinforcement learning agent is evaluated and it has shown room for improvement in various directions.

1 Introduction

Table turnover rate, measured by the amount of time a party occupies a table over a specific period of time, is a critical metric for restaurateurs as it has a direct impact on the bottom line. Some factors contributing to this include the restaurant size and layout, which may have bottlenecks at critical points, as well as staff efficiency in preparing food and clearing tables. Yet another factor, which we are trying address in this paper, is the amount of time a customer takes to place an order. This can be exacerbated by extensive menu options.

The adoption of self-ordering systems in restaurants has gained significant traction in recent years, offering patrons autonomy and convenience while streamlining operations for establishments and partially addressing the shortage of manpower problem faced in the industry. However due to limited screen real-estate on mobile QR self-ordering applications or the unnatural user interface of self ordering kiosks, customers often take more time browsing through the menu to familiarise themselves with its offerings or even to search for a particular item to place the order.

Singapore's IMDA recently rolled out a Retail Recommendation Engine (RRE) (IMDA, 2023) that covers F&B recommendation in an effort to improve table turnover rate as well, providing evidence towards a pain of the industry as well as a possible solution.

With aforementioned points in mind, coupled with the team's access to 11 months of actual customer order data, this paper explores a deep reinforcement learning based recommendation system for self-ordering kiosks and QR ordering applications, building upon the strength of the digital nature of those systems while addressing its weaknesses and ultimately improving table turnover rate by reducing decision time.

2 Literature Review

With the development and widespread of digital ordering platforms, an increasing demand for suitable and efficient recommendation system are rising (Yu et al., 2018). A good recommendation system could enhance the user experience, increase purchase rates, shorten the decision-making time for customers to improve store operating efficiency. As computational capabilities improves and new algorithms emerge, the algorithms for automated recommendation systems are also evolving. They can be broadly divided into two stages: traditional techniques based on non-reinforcement learning and reinforcement learning techniques.

2.1 Non-RL based Recommendation Techniques

The main idea of traditional recommendation approaches heavily depends on the context of item and user data. For content-based filtering (Pazzani and Billsus, 2007) (Mathew et al., 2016), the relationship between items is carefully considered. Similarity of the items is the primary basis of such approach. The performance of this method heavily depends on the description of the item and the alignment of the description and the user preferences. Later, collaborative filtering Schafer et al. (2007) (Herlocker et al., 2004) is able to take a bigger user pool into account. Similar users tend to prefer similar items is the main idea behind collaborative filtering. Matrix factorization (Koren et al., 2009) thus is proposed to address the problem of instability when calculating the similarity from sparse user data.

2.2 RL based Recommendation Techniques

In order to achieve better efficiency and effectiveness in terms of the recommendations, various methodologies have been explored by researchers in the realm of reinforcement learning for recommender systems. Model-free techniques lay the foundation of the implementation. For instance, with Q-learning (Watkins and Dayan, 1992) and SARSA (Rummery and Niranjan, 1994) providing straight forward methods, behaviors can be simulated by direct learning from actions. These methods, however, often suffers from complex environment or behaviors (Afsar et al., 2022). Provided detailed knowledge of the environment, techniques like dynamic programming (Demirezen and Kumar, 2016) or Monte Carlos (Rong et al., 2014) have been implemented for boosting the recommend accuracy, but such requirement for understanding the environment as well as heavy computational costs make them less practical for real-world application.

Deep reinforcement learning is thus introduced to enhance traditional reinforcement learning approach, seeking balance between the capability to manage complex behavior simulations and hardware consumption. Value-based methods such as Deep Q-Network often plays the foundation role of a model (Afsar et al., 2022). X Zhao (Zhao et al., 2018) enhances the basic DQN model by incorporating negative feedback alongside positive feedback. State representation was refined, and a network architecture was designed to handle positive and negative feedback separately. A pairwise regularization term is also introduced to differentiate between similar items. The idea of Q-value is further developed in Ie et al. (2019), showing that the long tern value of a recommender can be learned at the level of individual items.

On the other hand, instead of focusing on learning an optimal action-value function with one network from the environment, actor-critic algorithm (Konda and Tsitsiklis, 1999) was implemented for sequence prediction task (Bahdanau et al., 2016). An actor-critic model is the combination of two networks. One representing an actor that decides the action to take based on the policy while the other evaluates the actions using a value function. An off-policy actor-critic agent was scaled for industrial recommendation use case in Chen et al. (2022). In Liu et al. (2018), a special module named state representation module is introduced to enhance the modelling of the interaction between items and users.

3 Our Methodology

In this section, we would like to detail the approach for our recommendation system. It starts with the overall design of our work, followed by details of the data pipeline and reinforcement learning model.

3.1 Overall Design

Our proposed design for reinforcement learning-based recommendation system is demonstrated in Figure 1. Users access to our front-end which is a web application in order to login to their account and proceed to view and select the menu. When users login, we send a `/reset` API to reset the environment to the new user information. Upon arriving in the menu viewing page, our front-end sends a `/recommend` API call to receive the list of dish ID(s) which are the recommended items from the agent. Subsequently, users may choose (or not choose) any items in the recommendation, we will send another API call (`/feedback`) to reflect the user feedback to the agent. This feedback works as a trigger for the agent to move to the next state in the environment. When the user activities are finished (users finish their order, or they log out of the system), we will update the order database accordingly.

When the training is triggered, the data ingestion service retrieves the latest data from the database, pre-processes it by transforming the data to user-item matrix and learns the embeddings of both user and item and feed the results to the recommender. Finally, the recommender will start train the new agent from the latest embeddings.

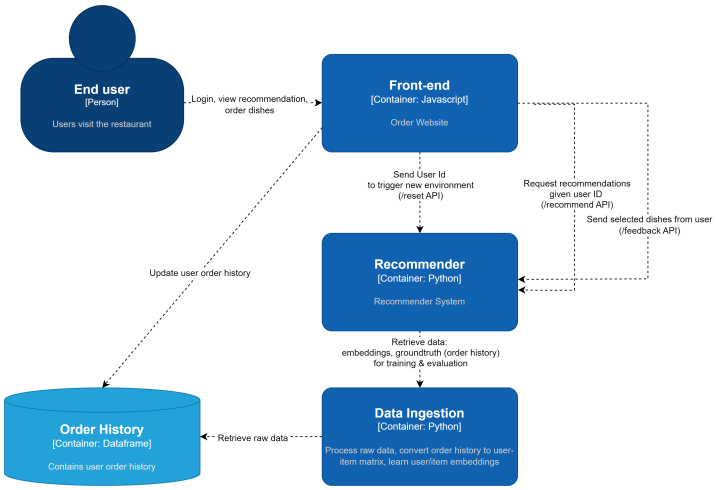
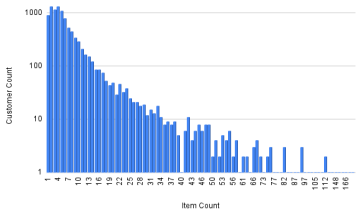


Figure 1: Overall design of the order recommender system.

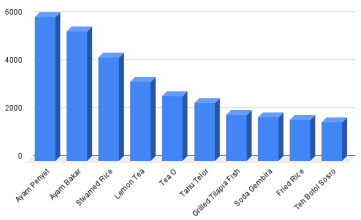
3.2 Data Pipeline

The source data we have access to is 11 months of customer order data from a particular chain of restaurants in Singapore. The menu data consists of over 150 menu items, however these include some internal test and seasonal items that are not consistent across the period. The menu structure consists of menus (such as Lunch and Dinner) which have varying categories of items (such as Mains, Sides, Drinks), which subsequently have groups of modifier items (such as sugar level or spiciness).

(a) Customer order history distribution by number of orders



(b) Customer order history distribution by number of items



(c) Top 10 items sold by order count

Figure 2: Exploratory data analysis on our data set

The distribution of count of customers with number of orders across these 11 months is shown in Figure 2a while that of count of customers with number of items is shown in Figure 2b. The Top 10 items sold across these months is shown in Figure 2c with the actual numbers wrangled to avoid revealing true sales figures.

The data ingestion service pulls the raw data from the database and extracts information for recommendation framework, namely user ID, item ID, timestamp (the time ordered is placed in the system). Next, it learns the user and item embeddings which serve as inputs for RL model training and inference. First, an user-item matrix is constructed where each element of the matrix is the number of times the user buys the particular item. Then, SVD, a matrix factorization method (Bokde et al., 2015), is applied to produce 2 matrices: (1) an N user embeddings $\{u_1, u_2, \cdots, u_N\} \in \mathbb{R}^D$ and (2) M item embedding $\{i_1, i_2, \cdots, i_M\} \in \mathbb{R}^D$ where N, M, D are the number of users, number of items, and dimensions of the embedding space, respectively. We apply the same strategy for both training and evaluation procedures.

3.3 Our Model

Markov Decision Process Modelling We formulate the recommendation problem as a Markov Decision Process (MDP) problem. Using the MDP framework, one needs to define the states, actions, transitions, the reward function and the discount factor, or $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$. We consider each user profile

as a separate environment that the agent needs to operate on. Then, the recommendation procedure is formulated as follow:

- **States** $\mathcal{S}$ A state represents the user historical preferences to the dishes. We consider historical positive items are those dishes the user purchased in the past.
- **Actions** $\mathcal{A}$. An action is represented by a continuous vector of D dimensions. The action is the weight to each component of the item embedding. Hence, the relevant score of the ith item is calculate by the dot product between the action vector and the item embedding: $s_i = i_i a^T$.
- **Transitions** $\mathcal{P}$. The transition model is constructed depending on whether the agent is running on online (deployment) or offline process. During the deployment, user feedback, i.e., whether there are any recommended items that the user choose to purchase, are used to create the next state of the environment. During offline process such as training and evaluation, part of the user historical data is used to construct the next state.
- **Reward** $\mathcal{R}$. The reward is directly calculated based on user feedback (or data groundtruths) given a list of recommended items from the agent.
- **Discount Factor** γ. The parameter $\gamma \in [0, 1]$ balances between the immediate reward and the future, long-term reward.

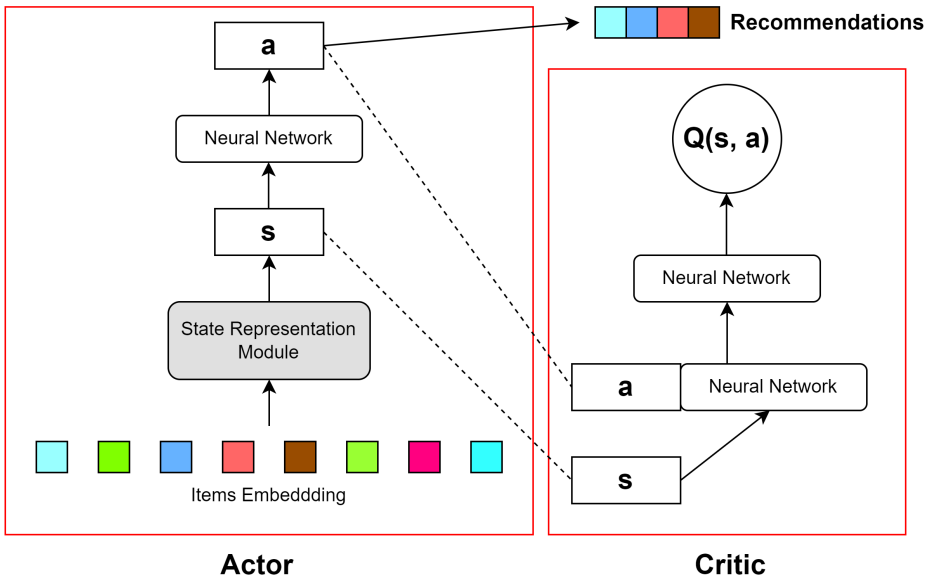


Figure 3: The Actor-Critic framework for recommendation agents.

The Actor-Critic Framework Our RL framework is illustrated in Figure 3. Followed what suggested in (Liu et al., 2018), we opt to use the Actor-Critic framework. As mentioned in Section 2.2, the Actor network π_θ works as a policy which creates a continuous action vector. It is optimized under the gradient of Q-value with respect to action vector: $\frac{\partial Q}{\partial a}$. The critic network Q_ω then evaluates the learned policy π by using $Q_\omega(s, a)$. In (Liu et al., 2018), the authors introduced 3 state representation modules $f(u, H)$. They are neural networks that learn the interaction between user embedding and positive item embeddings. We decided to choose DRR-Ave variant because DRR-Ave produces smaller number of dimension for state spaces.

Modelling state space One of the most important aspect of modelling a MDP problem is how to construct the state space. Given a specific user, we sort their order by timestamp from the oldest order to the most recent one (Figure 4). Then, we choose the first few orders as **historical orders** to construct

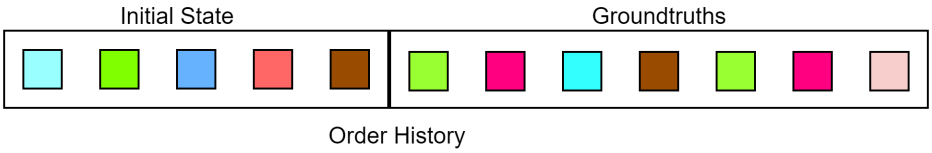


Figure 4: The users ordered 12 items, we use the first 5 items for the initial state while the remaining 7 are used for updating reward value and next states.

the initial state while the remaining orders are kept as the **groundtruths** for calculating rewards and updating the subsequent states. Given an user in the user set $i \in \{1, \cdots, N\}$, we call the groundtruth set of the ith user is G_i.

The reward function Instead of directly calculating the reward from action vector a and state vector s_t, we use the recommended item set P_t (inferred from action a and the item space) and the groundtruth set G_u of user u. An item $i \in P_t$ is considered as a positive if $i \in G_u$. Otherwise when $i \notin G_u$, it is a negative item.

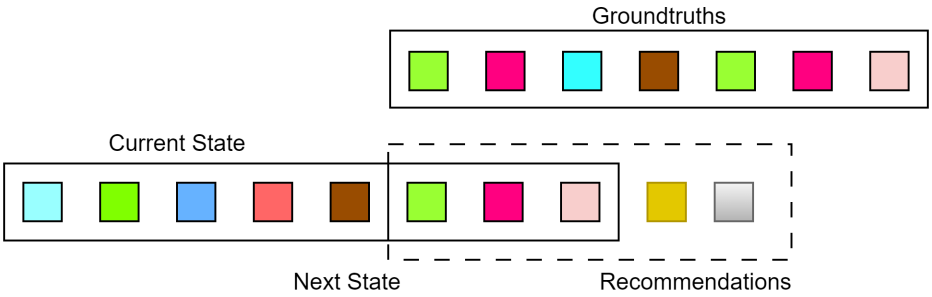


Figure 5: Although there are 5 recommendations, there are only 3 items matched with the groundtruth. Hence the reward of this recommendations is 2. The next state is constructed by concatenating the previous positive items with the new positive items.

The reward of P_t (or a_t) is calculated as follow:

$$\mathcal{R}(a_t, G_u) = R(P_t, G_u) = \sum_{i \in P_t} 1 \times [i \in G_u] + -0.5 \times [i \notin G_u] \tag{1}$$

where $[\cdot]$ is the Iversion bracket. An example is given in Figure 5.

Algorithm 1: Training procedure

Input : Historical data from users,
MDP parameters: discount factor γ, initial window size: n, Reward function $\mathcal{R}$.
T: maximum number of times the agent can recommend to the user.
k: Number of recommended items per each action.

Initialize the online Actor π_θ and Critic Q_ω network.
Initialize the target Actor $\pi_{\theta'}$ and Critic $Q_{\omega'}$ network: $\theta' = \theta$, $\omega' = \omega$.
Initialize the prioritized experience replay D.
for $i \in [1, \cdots, M]$ **do**
  Uniformly sample an user u.
  Create the initial historical set $H_0 = \{i_1, i_2, \cdots, i_n\}$.
  $t = 0$
  while $t < T$ **do**
    Create the state embedding: $s_t = f(u, H_t)$ where u is the user embedding.
    $a_t = \pi_\theta(s_t)$ with ϵ-greedy exploration.
    Recommend item set $P_t = \{i'_1, i'_2, \cdots, i'_k\}$ according to a_t
    $r_t = \mathcal{R}(P_t, G_u)$ according to Equation 1.
    Construct the next state by concatenating the current item set H_t with m positive items in P_t: $H_{t+1} = \{i_1, i_2, \cdots, i_n, i'_{j(1)}, \cdots, i'_{j(m)}\}$ where $j(.)$ is a 1-1 mapping from $\{1, \cdots, m\}$ to the index of positive items in P_t.
    $s_{t+1} = f(U_u, H_{t+1})$
    Push transition (s_t, r_t, a_t, s_{t+1}) to the replay buffer.
    Sample N transitions (s, a, r, s') from buffer D.
    $y = r + \gamma Q_{\omega'}(s', \pi_{\theta'}(s'))$
    Train the online Critic network Q_ω via the MSE loss: $L(y, (s, a)) = \frac{1}{N} \sum_i^N (y_i - Q_\theta(s_i, a_i))^2$
    Train the online Actor network π_θ via the gradient of Q-value and the action vector.
    Update the target Actor network: $\theta' = \tau\theta + (1 - \tau)\theta'$
    Update the target Critic network: $\omega' = \tau\omega + (1 - \tau)\omega'$
    $t = t + 1$
return π_θ *and* Q_ω

Training procedure We use the similar training procedure as described in (Liu et al., 2018) with a few tweaks, and our implementation is shown in Algorithm 1. Our algorithm recommends a set of k items instead of recommending a single item per step. Moreover, the reward function (Equation 1) is also different from (Liu et al., 2018) To create the next state, we append the positive recommended items into the existing positive list (see Figure 5) which includes both items in the initial state and positive items in subsequent steps.

4 Results and Discussions

In this section, we will first talk about details about the experiment setting, followed by the findings to assess the efficacy as a recommendation system as well as a reinforcement learning agent, and lastly discussion on further improvement to our work.

4.1 Experiment Setting

The dimension and item embedding dimension are fixed at 4. We experimented with different values of initial state windows, i.e, $[3, 5, 8, 10, 15]$ while during evaluation we fixed the value at 3. The value of k, number of recommended items per action, is set to 10.

There is a discussion in (Liu et al., 2018) regarding the impact of episode length T on the reinforcement learning agent. In our work, we did not experiment on this aspect due to the time limitation. We picked our episode length as 20 as suggested in (Liu et al., 2018). It means the training moves to the next episode when the agent recommends more than 20 times on one user.

The following setting has been chosen for our algorithm: the discount factor γ at 0.9, the buffer size of the prioritized experience replay at 10000. For ϵ-greedy exploration, $\epsilon = 1.0$ with decay rate at 10^{-6} to encourage the agent explore more at early stages on the learning. For the Actor network, the number of hidden dimension is 128 , while that for the Critic network is set to 32. To train all networks in the framework, we use Adam optimizer (Kingma and Ba, 2014) and the number of episodes per training is 5000.

4.2 Experiment Finding

Initial State Size	Precision@5	Recall@1	Average reward per episode
3	0.032	0.147	-88.9
5	0.034	0.164	-88.2
8	0.009	0.045	-91.1
10	0.023	0.106	-88.7
15	0.017	0.077	-87.4

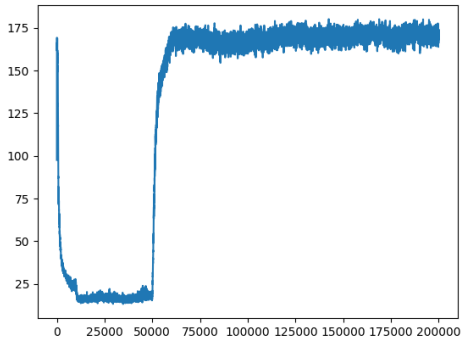
Table 1: Evaluation metrics (Precision@5 and Recall@1) and average rewards per episode at different initial state size

As mentioned in Section 4.1, we repeated the training with different values of initial state windows and the results of the evaluation metrics as well as the average reward per episode is shown in Table 1. Our best model performance is found at shorter initial state size and it decreases as the initial state size grows. One possible explanation is that our model suffers from lack of data (see Figure 2a).

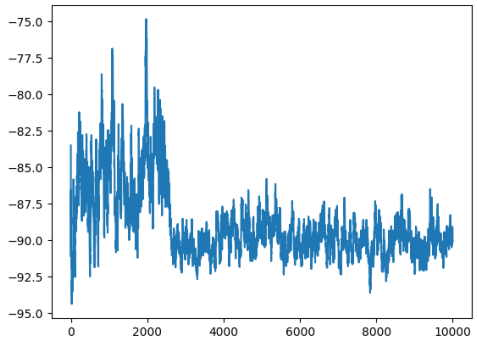
User historical orders	Tahu Telor, Duo Ayam Penyet, Soda Gembira, Duo Ayam Bakar
Subsequent orders	Tahu Telor, Soda Gembira
Recommended items@5	Tahu Telor, Mineral Water, Ayam Penyet Giant Leg, Belinjau Cracker, Omelette
Precision@5	0.2
Recall@1	1

Table 2: Example result of a successful recommendation, 1 step evaluation

Table 2 shows an example of our recommendation and its evaluation when initial state size is 5 for a random user. One of the recommended item, Tehu Telor, appears in the user historical selection and it is considered a successful recommendation as it is also found in the user's subsequent orders, which is used as ground truth for evaluation calculation. Precision@5 and Recall@1 is calculated for this selected user.



(a) Critic loss per episode (smoothed), initial state size = 3



(b) Total rewards per episode (smoothed), initial state size =3

Figure 6: Training critic loss and total reward per episode

Figure 6a and Figure 6b are examples graphs of the critic loss per episode and total reward per episode during our training. Observing Figure 6b, the algorithm looks trapped after certain episode and struggling to find the best strategy to achieve high rewards. Recall the discussion in (Liu et al., 2018)

regarding the episode length T for exploration and exploitation dilemma, our model may suffer from too much exploitation. In other words, the algorithm is trapped by a sub-optimal strategy, unable to get out from the situation and retrieve a better strategy.

The result shows that we have a lot of rooms to improve for future iteration. A reasonable benchmark from (Liu et al., 2018) is 0.3917 Precision@5 using the same model structure and Yahoo!Music data. This is about than 10 times compare to our best Precision@5 achieved.

4.3 Discussion

In the last section, we would like to discuss several possible ways to improve our reinforcement learning model further.

Item Embedding Item embedding is the input of the our recommendation and we cannot emphasize enough the importance of this input. As elaborated in Section 3.2, our item embedding method is SVD, a simple way to extract information from the items without too much semantic meaning is taken into consideration. While the original implementation in (Liu et al., 2018), is using a PMF approach, the same as in (Mnih and Salakhutdinov, 2007). This approach is known to work well on large, sparse and imbalance dataset and it is one of the potential improvements that we can consider.

Another improvement related to item embedding, is to enrich the data input by adding more dimensions. In our current approach, we have only take the dish ID into account when constructing the item embedding. To enrich the data input, we can add category of the dish (such as starter, main course or drinks) or innate attributes of the dish (such as ingredients, flavor etc.) as mentioned in 3.2.

Reward Function Design Compared to (Liu et al., 2018), our reward function is much more simpler due to the fact that data other than transaction history is difficult to obtain because they may be on recorded on different platforms or simply not available. As a result, our user interaction data is very limited as the transaction data is usually sparse. The probability of transiting to the next state becomes very low and it is difficult for the model to learn as it is difficult to locate the actual rewards.

One simple idea is to magnify the significance of the positive reward by making the positive reward much higher (e.g. 100 times more) than the negative ones. In addition, we can improve the reward function further by collecting more user interaction signals such as clicks, scroll time as well as add to cart signal and include these in the reward calculation. All of these signals can be considered as either positive signals as well as negative signals as suggest in (Zhao et al., 2018), indicating the user's interest positively or negatively.

Training Fine Tuning As established in Section 3.3, our model has a lot of parameters and settings that can be experimented with. To name a few, popular suspects for further tuning include the learning rate α, the soft replacement coefficient τ, batch size, the episode length T and so on. Following our findings in Section 4.2, one immediate action is to experiment on different episode length T for a grid of values less than 20, such as [3,5,8,12,15], in order to search for optimal performance that balance the exploration-exploitation dilemma.

In addition, other components of the model are also possible candidates to explore. One example is our exploration strategy, ϵ-greedy, can be replaced by a variety of other deep reinforcement learning strategies as in (Weng, 2020).

5 Conclusion

In this project, we have examined different reinforcement learning papers in order to find a reasonable solution for the real world problem which is to recommend for restaurant orders. Then we proposed our method to build a recommendation system by following closely to the paper (Liu et al., 2018), which we find most closely related to our situation, with some adaptation. Despite the result is not ideal, we have proposed several directions that we can potentially work towards if we were given longer time to explore the topic. Our demonstration video can be accessed at https://youtu.be/4aZwodtBoAg. [1] We also release the agent implementation at: https://github.com/CS5446-BCKR/RLRS.

[1] Disclaimer: All data and images in the linked video and this report are for academic purpose only and not for reproduction.

References

Afsar, M. M., Crump, T., and Far, B. (2022). Reinforcement learning based recommender systems: A survey. *ACM Computing Surveys*, 55(7):1–38.

Bahdanau, D., Brakel, P., Xu, K., Goyal, A., Lowe, R., Pineau, J., Courville, A., and Bengio, Y. (2016). An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv:1607.07086*.

Bokde, D., Girase, S., and Mukhopadhyay, D. (2015). Matrix factorization model in collaborative filtering algorithms: A survey. *Procedia Computer Science*, 49:136–146.

Chen, M., Xu, C., Gatto, V., Jain, D., Kumar, A., and Chi, E. (2022). Off-policy actor-critic for recommender systems. In *Proceedings of the 16th ACM Conference on Recommender Systems*, pages 338–349.

Demirezen, E. M. and Kumar, S. (2016). Optimization of recommender systems based on inventory. *Production and Operations Management*, 25(4):593–608.

Herlocker, J. L., Konstan, J. A., Terveen, L. G., and Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53.

Ie, E., Jain, V., Wang, J., Narvekar, S., Agarwal, R., Wu, R., Cheng, H.-T., Lustman, M., Gatto, V., Covington, P., et al. (2019). Reinforcement learning for slate-based recommender systems: A tractable decomposition and practical methodology. *arXiv preprint arXiv:1905.12767*.

IMDA (2023). Accelerate business growth in retail and f&b with the power of a.i. https://www.imda.gov.sg/about-imda/emerging-technologies-and-research/artificial-intelligence/recommendation-engine. Accessed: 2024-04-20.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Konda, V. and Tsitsiklis, J. (1999). Actor-critic algorithms. In Solla, S., Leen, T., and Müller, K., editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press.

Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37.

Liu, F., Tang, R., Li, X., Ye, Y., Chen, H., Guo, H., and Zhang, Y. (2018). Deep reinforcement learning based recommendation with explicit user-item interactions modeling. *CoRR*, abs/1810.12027.

Mathew, P., Kuriakose, B., and Hegde, V. (2016). Book recommendation system through content based and collaborative filtering method. In *2016 International conference on data mining and advanced computing (SAPIENCE)*, pages 47–52. IEEE.

Mnih, A. and Salakhutdinov, R. R. (2007). Probabilistic matrix factorization. In Platt, J., Koller, D., Singer, Y., and Roweis, S., editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc.

Pazzani, M. J. and Billsus, D. (2007). Content-based recommendation systems. In *The adaptive web: methods and strategies of web personalization*, pages 325–341. Springer.

Rong, Y., Wen, X., and Cheng, H. (2014). A monte carlo algorithm for cold start recommendation. In *Proceedings of the 23rd international conference on World wide web*, pages 327–336.

Rummery, G. A. and Niranjan, M. (1994). *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK.

Schafer, J. B., Frankowski, D., Herlocker, J., and Sen, S. (2007). Collaborative filtering recommender systems. In *The adaptive web: methods and strategies of web personalization*, pages 291–324. Springer.

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

Weng, L. (2020). Exploration strategies in deep reinforcement learning. *lilianweng.github.io*.

Yu, C., Tang, Q., Liu, Z., Dong, B., and Wei, Z. (2018). A recommender system for ordering platform based on an improved collaborative filtering algorithm. In *2018 International Conference on Audio, Language and Image Processing (ICALIP)*, pages 298–302. IEEE.

Zhao, X., Zhang, L., Ding, Z., Xia, L., Tang, J., and Yin, D. (2018). 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.