Delayed Rewards in the context of Reinforcement Learning based Recommender Systems

Debmalya Biswas[1]

Abstract. We present a Reinforcement Learning (RL) based approach to implement Recommender systems. The results are based on a real-life Wellness app that is able to provide personalized health related content to users in an interactive fashion. Unfortunately, current recommender systems are unable to adapt to continuously evolving features, e.g. user sentiment, and scenarios where the RL reward needs to be computed based on multiple and unreliable feedback channels (e.g., sensors, wearables). To overcome this, we propose three constructs: (i) weighted feedback channels, (ii) delayed rewards, and (iii) rewards boosting, which we believe are essential for RL to be used in Recommender Systems. Finally, we also provide some implementation details on how the Wellness App based on Azure Personalizer was extended to accommodate the above RL constructs.

1 INTRODUCTION

Wellness apps have historically suffered from low adoption rates. Personalized recommendations have the potential of improving adoption, by making increasingly relevant and timely recommendations to users. While recommendation engines (and consequently, the apps based on them) have grown in maturity, they still suffer from the 'cold start' problem and the fact that it is basically a push-based mechanism lacking the level of interactivity needed to make such apps appealing to millennials.

We present a Wellness app case-study where we applied a combination of Reinforcement Learning (RL) and Natural Language Processing (NLP)/Chatbots to provide a highly personalized and interactive experience to users. We focus on the interactive aspect of the app, where the app is able to profile and converse with users in real-time, providing relevant content adapted to the current sentiment and past preferences of the user.

The core of such chatbots is an intent recognition Natural Language Understanding (NLU) engine [9], which is trained with hard-coded examples of question variations. When no intent is matched with a confidence level above 30%, the chatbot returns a fallback answer. The user sentiment is computed based on both the (explicit) user response and (implicit) environmental aspects, e.g. location (home, office, market, ...), temperature, lighting, time of the day, weather, other family members present in the vicinity, and so on; to further adapt the chatbot response.

[1] Philip Morris Products S. A., Lausanne, Switzerland, email: debmalya.biswas@pmi.com

RL refers to a branch of Artificial Intelligence (AI), which is able to achieve complex goals by maximizing a reward function in real-time. The reward function works similar to incentivizing a child with candy and spankings, such that the algorithm is penalized when it takes a wrong decision and rewarded when it takes a right one – this is reinforcement. The reinforcement aspect also allows it to adapt faster to real-time changes in user sentiment. For a detailed introduction to RL frameworks, the interested reader is referred to [3].

Previous works have explored RL in the context of Recommender Systems [5, 7, 10], and enterprise adoption also seems to be gaining momentum with the recent availability of Cloud APIs (e.g. Azure Personalizer [2, 6]) and Google's RecSim [1]. However, they still work like a typical Recommender System. Given a user profile and categorized recommendations, the system makes a recommendation based on popularity, interests, demographics, frequency and other features. The main novelty of these systems is that they are able to identify the features (or combination of features) of recommendations getting higher rewards for a specific user; which can then be customized for that user to provide better recommendations. Unfortunately, this is still inefficient for real-life systems which need to adapt to continuously evolving features, e.g. user sentiment, and where the reward needs to computed based on multiple and unreliable feedback channels (e.g., sensors, wearables).

The rest of the paper is organized as follows: Section 2 outlines the problem scenario and formulates it as an RL problem. In Section 3, we propose three RL constructs needed to overcome the above limitations: (i) weighted feedback channels, (ii) delayed rewards, and (iii) rewards boosting, which we believe are essential constructs for RL to be used in Recommender Systems. 'Delayed Rewards' in this context is different from the notion of Delayed RL [8], where rewards in the distant future are not considered as valuable as immediate rewards. In our notion of 'Delayed Rewards', a received reward is only applied after its consistency has been validated by a subsequent action. We provide implementation details in Section 4, on how to extend an RL powered Wellness App based on Azure Personalizer, to accommodate the above constructs. Section 5 concludes the paper providing some directions for future work.

2 PROBLEM SCENARIO

In this section, we set the problem context and formulate it as a Reinforcement Learning problem.

2.1 Wellness App

The Wellness app supports both push based notifications, where personalized health, fitness, activity, etc. related recommendations are

pushed to the user; as well as interactive chats where the app reacts in response to a user query. We assume the existence of a knowledgebase KB of articles, pictures and videos, with the artifacts ranked according to their relevance to different user profiles / sentiments.

The Wellness app architecture is described in Fig. 1, which shows how the user response and environmental conditions are:

1. gathered using available sensors to compute the 'current' feedback, including environmental context (e.g. webcam pic of the user can be used to infer the user sentiment to a chatbot response / notification, the room lighting conditions and other users present in the vicinity),
2. which is then combined with the user conversation history to quantify the user sentiment curve and discount any sudden changes in sentiment due to unrelated factors;
3. leading to the aggregate reward value corresponding to the last chatbot response / app notification provided to the user.

This reward value is then provided as feedback to the RL agent, to choose the next optimal chatbot response / app notification from the knowledgebase. It is worthwhile noting here that capturing the user sentiment, esp. the environmental aspects, requires a high degree of knowledge regarding the user context. As such, we need to perform this in a privacy preserving fashion. We suffice to say here that appropriate privacy protections are provided by the 'Privacy' block in Fig. 1, and further details are provided in [4].

2.2 RL Formulation

We formulate the RL Engine for the above scenario as follows (illustrated in Fig. 2):

- Action (a): An action a in this case corresponds to a KB article which is delivered to the user either as a push notification, or in response to a user query, or as part of an ongoing conversation.
- Agent (A): is the one performing actions. In this case, the Agent is the App delivering actions to the users, where an action is selected based on its Policy.
- Policy (π): is the strategy that the agent employs to select the next best action. Given a user profile U_p, (current) sentiment U_s, and query U_q; the Policy function computes the product of the article scores returned by the NLP and Recommendation Engines respectively, selecting the item with the highest score as the next best action:
  - The NLP Engine (NE) parses the query and outputs a score for each KB article, based on the "text similarity" of the article to the user query.
  - Similarly, the Recommendation Engine (RE) provides a score for each article based on the reward associated with each article, with respect to the user profile and sentiment. The Policy function π can be formalized as follows:

$$\pi(U_p, U_s, U_q) = a \mid \max_a [NE(a, U_q) \times RE(a, U_p, U_s)] \tag{1}$$

- Reward (r): refers to the feedback by which we measure the success or failure of an agent's recommended action. The feedback can e.g. refer to the amount of time that a user spends reading a recommended article. We consider a 2-step reward function computation where the feedback f_a received with respect to a recommended action is first mapped to a sentiment score, which is then mapped to a reward.

$$r(a, f_a) = s(f_a) \tag{2}$$

where r and s refer to the reward and sentiment functions, respectively. Once computed, the KB is updated with the computed reward / sentiment for the corresponding action.

3 RL REWARD AND POLICY EXTENSIONS

In this section, we show how the Reward and Policy functions are extended to accommodate the real-life challenges posed by our RL based Wellness App.

3.1 Weighted (Multiple) Feedback Channels

As described in Fig. 1, we consider a multi-feedback channel, with feedback captured from user (edge) devices / sensors, e.g. webcam, thermostat, smartwatch, or a camera, microphone, accelerometer embedded within the mobile device hosting the app. For instance, a webcam frame capturing the facial expression of the user, heart rate provided by the user smartwatch, can be considered together with the user provided text response "Thanks for the great suggestion"; in computing the user sentiment to a recommended action.

Let $\{f_{a_1}, f_{a_2}, ...f_{a_n}\}$ denote the feedback received for action a. Recall that $s(f)$ denotes the user sentiment computed independently based on the respective sensory feedback f. The user sentiment computation can be considered as a classifier outputting a value between 1-10. The reward can then be computed as a weighted average of the sentiment scores, denoted below:

$$r_a(\{f_{a_1}, f_{a_2}, ...f_{a_n}\}) = \sum_{i=1}^{n} (w_i \times s(f_{a_i})) \tag{3}$$

where the weights $\{w_{a1}, w_{a2}, ...w_{an}\}$ allow the system to harmonize the received feedback, as some feedback channels may suffer from low reliability issues. For instance, if f_i corresponds to a user typed response, f_j corresponds to a webcam snapshot; then higher weightage is given to f_i. The reasoning here is that the user might be 'smiling' in the snapshot, however the 'smile' is due to his kid entering the room (also captured in the frame), and not necessarily in response to the received recommendation / action. At the same time, if the sentiment computed based on the user text response indicates that he/she is 'stressed', then we give higher weightage to user explicit (text response) feedback in this case.

3.2 Delayed Rewards

A 'delayed rewards' strategy is applied in the case of reward inconsistency, where the current (computed) reward is 'negative' for an action to which the user has been known to react positively in the past; or vice versa. For instance, let us consider that the user sentiment is low for a recommendation of category 'Shopping', to which the user has been known to react very positively (to other 'Shopping' related recommendations) in the past. Given such inconsistency, the delayed rewards strategy buffers the computed reward r_{at} for action a_t at time t; and provides an indication to the RL Agent-Policy (π) to try another recommendation of the same type ('Shopping') - to validate the user sentiment - before updating the rewards for both a_t and a_{t+1} at time $t+1$.

To accommodate the 'delayed rewards' strategy, the rewards function is extended with a memory buffer that allows the rewards of last m actions $[a_{t+m}, a_{t+m-1}, ..., a_t]$ to be aggregated and applied

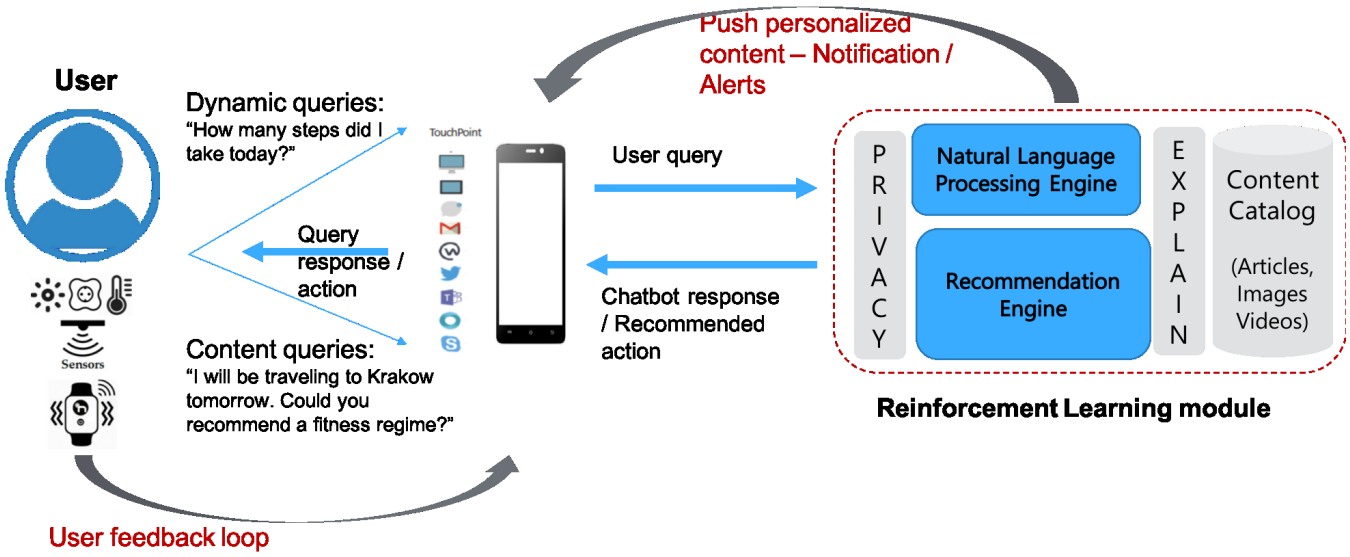

Figure 1. Wellness app architecture

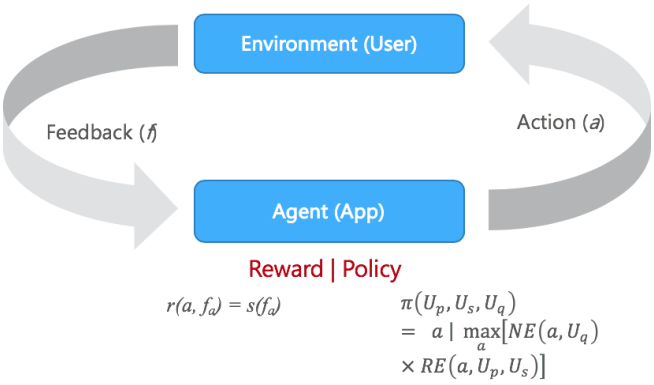

Figure 2. RL formulation

retroactively at time $(t + m)$. The delayed rewards function dr is denoted as follows:

$$dr_{a_{ti} \in \{a_t, a_{t+1}, \ldots, a_{t+m}\}} \mid (t+m) = \sum_{i=0}^{m} (w_i \times r_{a_{ti}}) \quad (4)$$

where $\mid \; t + m$ implies that the reward for the actions $[a_{t+m}, a_{t+m-1}, ..., a_t]$, although computed individually; can only be applied at time $(t + m)$. As before, the respective weights w_i allow us to harmonize the effect of an inconsistent feedback, where the reward for an action a_{ti} is applied based on the reward computed for a later action $a_{(t+1)i}$.

To effectively enforce the 'delayed rewards' strategy, the Policy π is also extended to recommend an action of the same type, as the previous recommended action; if the delay flag d is set $(d = 1)$: The "delayed" Policy π_{dt} is outlined below:

$$\pi_{dt}(U_p, U_s, U_q) = \begin{cases} d = 1: & a_t \mid r_{a_t} \approx r_{a_{t-1}} \\ d = 0: & a \mid \max_a [NE(a, U_q) \times RE(a, U_p, U_s)] \end{cases} \quad (5)$$

The RL formulation extended with delayed reward / policy is illustrated in Fig. 3.

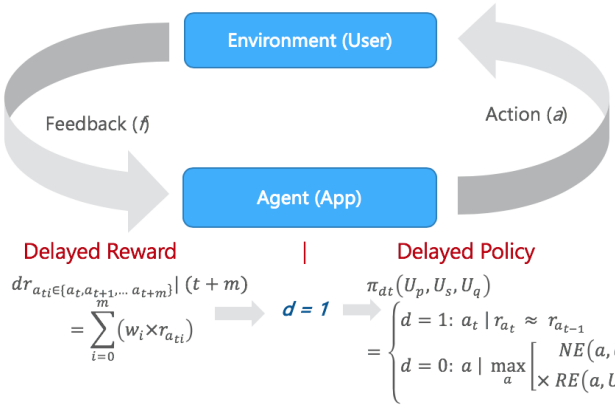

Figure 3. Delayed Reward - Policy based RL formulation

3.3 Rewards Boosting

Rewards boosting, or rather rewards normalization, applies mainly to continuous chat interactions. In such cases, if the user sentiment for a recommended action is 'negative'; it might not be the fault of the last action only. It is possible that the conversation sentiment was already degrading, and the last recommended action is simply following the downward trend. On the other hand, given a worsening conversation sentiment, a 'positive' sentiment for a recommended action implies

that it had a very positive impact on the user; and hence its corresponding reward should be boosted. For example, let us consider a ranking of the user sentiments:

Disgusted → Angry → Sad → Confused → Calm → Happy

Given this, a change from 'Disgusted' to 'Happy' would lead to a much higher (positive) boost, than a (negative) change from 'Confused' to 'Sad'.

The boosted reward rb_{at} for an action a_t at time t is computed as follows:

$$rb_{a_t} = \frac{1}{2}(r_{a_t} - rb_{a_{t-1}}) \times r_{a_t} \quad (6)$$

It is easy to see that a 'positive' $r_{a_t} = 7$ following a 'negative' $rb_{a_{t-1}} = -5$, will lead to r_{a_t} getting boosted by a factor of $\frac{1}{2}(7 - (-5)) = 6$. On the same lines, a 'negative' $r_{a_t} = -6$ following a 'positive' $rb_{a_{t-1}} = 4$, will lead to r_{a_t} getting further degraded by a factor of $\frac{1}{2}(-6 - 4) = -5$.

We leave it as future work to extend the 'boost' function to last n actions (instead of just the last action above). In this extended scenario, the system maintains a sentiment curve of the last n actions, and the deviation is computed with respect to a curve, instead of a discrete value. The expected benefit here is that it should allow the system to react better to user sentiment trends.

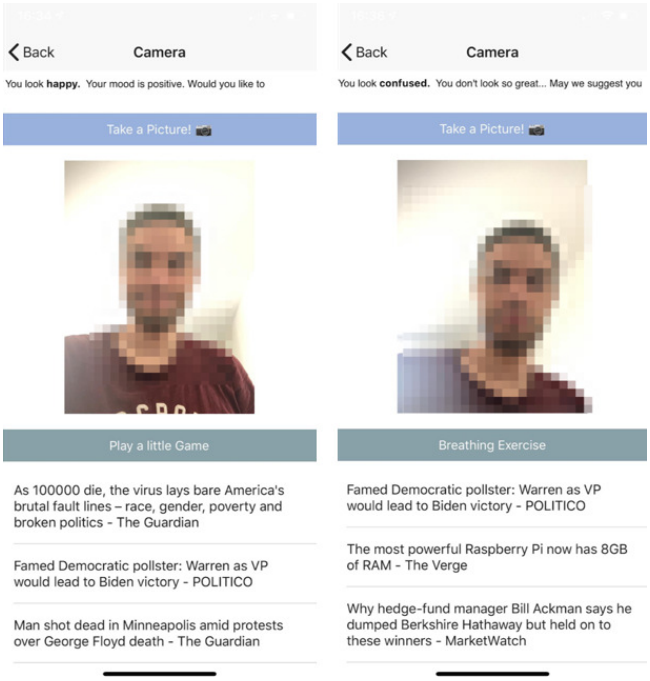

Figure 4. Wellness Recommender App screenshots

4 IMPLEMENTATION

In this section, we extended our RL powered Wellness App based on Azure Personalizer, to accommodate the constructs outlined in the previous section. Azure Personalizer [2] is a Cloud based API providing an implementation of RL Contextual Bandits [6]. In short, Personalizer provides two primary APIs:

- Rank API: The mobile app invokes Rank API with a list of actions and their features, and the user context and features. Given this, the Rank API returns a list of ranked actions. Internally, the Personalizer app uses the Explore-Exploit trade-off to rank the actions:
  - Exploit: Ranks the actions based on past data (current inference model).
  - Explore: Select a different action instead of the top action. The 'explore' percentage is a configurable parameter, and can be set along the lines of an epsilon greedy strategy.
- Rewards API: The mobile app presents the ranked content (returned by Rank API) and computes the reward corresponding to each action. It then invokes the Reward API to return the computed rewards to Personalizer. Personalizer correlates the action-reward, updating its inference model.

We now provide details of our Wellness Recommender App (illustrated in Fig. 4). In addition to the usual article / activity recommendations, the app provides the option to have a video chat as well to improve the 'interactive quotient' of the app. The implementation details of the RL constructs proposed in this paper are outlined below:

- *Multiple feedback channels*: in this case correspond to the live video feed and user interaction with an article / activity. Given a user snapshot (captured from the live feed), the sentiment score is computed using Azure Face API. The article / activity recommended by the app depends on both the article / activity relevance score and the 'current' user sentiment, e.g. tragic, sad, depressing, etc. related articles / activities are not shown unless the user is in a 'happy' mood (Fig. 4).
  Personalizer leaves the reward computation on the client side. We developed a Rewards Computation module (with reference to Eq. 3) that combines (i) the activity / article related score, i.e. the activity / app selected and the time spent interacting with it, together with (ii) the sentiment score computed based on the user snapshot; with a higher weightage assigned to the latter given its effectiveness in capturing the user reaction to a recommended article / activity.
- *Rewards boosting*: To accommodate this, we consider a ranking of the user sentiments returned by the Face API:
  Disgusted → Angry → Sad → Confused → Calm → Happy
  The rewards boosting factor (Eq. 6) is assigned proportional to the change in user sentiment before and after displaying the recommended activity / article.
- *Delayed rewards*: is a core RL construct that requires updating the backend Recommendation Engine and RL Reward and Policy functions (Eq. 4 and Eq. 5). As such, it is difficult to implement based on a Cloud API (without having direct access to the underlying Recommendation and RL Engines). We are in the process of adapting an Open Source Contextual Bandits implementation to provide the full 'delayed rewards' strategy.
  For now, we only simulated the behavior of the RL Reward function (Eq. 4) on the client (app) side. We added a memory buffer to the Rewards Computation module to store rewards computed for an iteration, until a 'similar' reward gets computed for a set of activities / articles with similar features recommended as part of another iteration. At this point, the aggregate rewards are returned to Personalizer (via Reward API) for recommended activities / articles of both iterations. This ensures that only consistent rewards are considered while training the Personalizer RL infer-

ence model.

5 CONCLUSION

In this work, we considered the implementation of a RL based Recommender System, in the context of a real-life Wellness App. RL is a powerful primitive for such problems as it allows the app to learn and adapt to user preferences / sentiment in real-time. However, during the case-study, we realized that current RL frameworks lack certain constructs needed for them to be applied to such Recommender Systems.

To overcome this limitation, we introduced three RL constructs that we implemented for our Wellness app: (i) weighted feedback channels, (ii) delayed rewards, and (iii) rewards boosting. The proposed RL constructs are fundamental in nature as they impact the interplay between Reward and Policy functions; and we hope that their addition to existing RL frameworks will lead to increased enterprise adoption.

ACKNOWLEDGEMENTS

I would like to thank Louis Beck and Sami Ben Hassan for their insights and support in developing the Wellness Recommender App.

REFERENCES

[1] Google RecSim, 2020 (accessed December 9, 2020). https://opensource.google/projects/recsim.
[2] Microsoft Azure Personalizer, 2020 (accessed December 9, 2020). https://azure.microsoft.com/en-us/services/cognitive-services/personalizer/.
[3] A. Barto and R. S. Sutton, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 2018.
[4] D. Biswas, 'Privacy Preserving Chatbot Conversations', in *Proceedings of the 3rd IEEE Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, (2020).
[5] S. Choi, H. Ha, U. Hwang, C. Kim, J. Ha, and S. Yoon. Reinforcement Learning based Recommender System using Biclustering Technique, 2018. arXiv:1801.05532.
[6] L. Li, W. Chu, J. Langford, and R. E. Schapire, 'A Contextual-Bandit Approach to Personalized News Article Recommendation', in *Proceedings of the 19th International Conference on World Wide Web (WWW)*, p. 661–670, (2010).
[7] F. Liu, R. Tang, X. Li, Y. Ye, H. Chen, H. Guo, and Y. Zhang. Deep Reinforcement Learning based Recommendation with Explicit User-Item Interactions Modeling, 2019.
[8] N. J. Nilsson. Delayed-Reinforcement Learning, 2020 (accessed December 9, 2020). http://heim.ifi.uio.no/ mes/inf1400/COOL/REF/Standford/ch11.pdf.
[9] E. Ricciardelli and D. Biswas, 'Self-improving Chatbots based on Reinforcement Learning', in *Proceedings of the 4th Multidisciplinary Conference on Reinforcement Learning and Decision Making (RLDM)*, (2019).
[10] N. Taghipour, A. Kardan, and S. S. Ghidary, 'Usage-Based Web Recommendations: A Reinforcement Learning Approach', in *Proceedings of the ACM Conference on Recommender Systems (RecSys)*, p. 113–120, (2007).