Q-Networks with Dynamically Loaded Biases for Personalization

1st Ján Magyar
DCAI, Technical University of Košice
Košice, Slovakia
jan.magyar@tuke.sk

2nd Peter Sinčák
DCAI, Technical University of Košice
Košice, Slovakia
peter.sincak@tuke.sk

2021 IEEE 19th World Symposium on Applied Machine Intelligence and Informatics (SAMI) | 978-1-7281-8053-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/SAMI50585.2021.9378651

***Abstract*—Personalization is ever more prevalent in digital systems in various application domains. Reinforcement learning is a method often applied to adjust a system's behavior to the user's preferences, but there are a number of hurdles when applying it in this context. We propose a novel neural network architecture for reinforcement learning agents specifically tailored to support personalization – Dynamically Loaded Biases Q-Network. We test our architecture on two environments simulating a personalization task and show that it can simultaneously learn a general behavior and adjust it to different environments.**

***Index Terms*—deep learning, multi-agent reinforcement learning, personalization, reinforcement learning, Q-network**

I. Introduction

Humans interact with multiple digital systems on a daily basis, and although these systems offer identical functionality to every user, their behavior is often adjusted to an individual. Generally, we can define such *personalization* as "a process that changes the functionality, interface, information access and content, or distinctiveness of a system to increase its personal relevance to an individual or a category of individuals" [1]. Personalization became widespread first in recommendation systems and web services, but benefits of personalized system behavior have been identified in further application domains, e.g. medicine [2], education [3], and human–computer interaction [4].

In recent years, reinforcement learning (RL) has gotten into the forefront of research into personalization using artificial intelligence. RL is an on-line learning paradigm that enables systems to learn during deployment. However, lack of training data poses a problem for the successful application of RL for personalization, given that sample efficiency is a key issue, and it is often hard or near-impossible to gather the necessary amount of training data solely from interactions with users.

In this paper we propose a novel architecture of neural nets called Dynamically Loaded Biases Q-Network meant to support sample efficiency and enable the network to not only personalize its behavior, but also learn a generally applicable policy from interactions. We test our solution in a simulated environment that has the specific characteristics of personalization problems, but abstracts away from their complexity. We compare our solution to an analogous deep Q-network, which is an often used RL approach in personalization problems. We show that our solution is able to acquire personalized behaviors in simulated personalization.

The paper is structured as follows: Section II formally defines reinforcement learning; Section III provides a short overview of recent research on reinforcement learning in personalization tasks and applicable techniques from multi-agent reinforcement learning; we introduce our proposed architecture in Section IV; in Section V we describe the experimental setup; Section VI presents our results, a discussion of which follows in Section VII; Section VIII concludes the paper and highlights open issues and possible future work.

II. Reinforcement learning

Reinforcement learning addresses problems in the framework of Markov decision processes (MDPs), where an agent – trained by the learning algorithm – interacts with an environment [5]. Throughout this interaction, the agent receives some reward that represents the suitability of choosing a given action in a given situation.

An MDP is defined as a tuple $< S, A, r, t >$, where S is the state space defining all possible states of the environment; A is the action space defining all actions available to the agent; $r : S \times A \to R$ is the reward function that assigns a numerical value to a state-action combination; and $t : S \times A \to T$ is the transition function defining the probability of the environment transitioning into a new state when the agent takes a particular action.

The goal of the agent is then to find a policy π that maximizes the expected sum of discounted rewards over the interaction. This accumulated reward is also called *value* and can be calculated as

$$v(s, \pi) = \sum_{t=0}^{\infty} \gamma^t E(r_t | \pi, s_0 = s) \tag{1}$$

where s_0 is the initial state, r_t is the reward at time step t and $\gamma \in [0, 1)$ is the discount factor. This discount factor is used to balance the trade-off between maximizing for immediate rewards against long-term optimal decision making. The higher the discount factor, the more preferred value is over reward.

Methods of reinforcement learning can enable an agent to learn about the environment either through the reward function and the state transition function (model-based reinforcement learning), or it can learn directly from interacting with the

environment (model-free reinforcement learning). In many applications, especially in the case of personalization, it is impossible to model the environment (the human). With such an approach, the agent is not aware of the reward and transition functions, it only approximates the value of action $a \in A$ taken in state $s \in S$ through its utility. One of the most widespread model-free algorithms is Q-learning [6].

In Q-learning the agent updates the Q-value representing the utility of action selection for each state-action pair based on Equation 2:

$$Q_{t+1}(s,a) = (1-\alpha_t)Q_t(s,a) + \alpha_t \left[r_t + \gamma \underset{a'}{max} Q_t(s',a') \right] \tag{2}$$

where $\alpha_t \in [0,1)$ is the learning rate that must decay over time so that the learning algorithm converges. During the training process, however, an agent that always selects the action with the highest utility is prone to getting stuck in a sub-optimal policy. This can be prevented by having the agent select the action with the highest expected Q-value most of the time, but a random action with a certain probability ε. By adapting this probability we can further improve the convergence of the learning algorithm.

Q-learning will converge if an agent acquires a large amount of experience for each $s \in S$. This means that as the size of the state space grows, Q-learning becomes less and less efficient. To tackle this problem, Q-networks which approximate the Q-function with a trainable neural network were introduced. Q-networks, deep Q-network (DQN) in particular [7], allow agents to work with continuous state spaces. The input of Q-networks is the state representation (often visual in the case of DQNs), and they predict the expected Q-values for each action in the output layer (the number of output nodes is equal to the size of A).

III. Reinforcement learning for Personalization

Methods of RL have been successfully applied in a number of personalization scenarios. The reward signal necessary to train agents and acquire better policies can be determined by the users themselves, or inferred from interaction or observing user behavior [8]. However, RL agents usually learn in a trial-and-error way, which is not always suitable for particular application domains where the cost of error might be too high, as was pointed out in [9]. This problem might be mitigated by the use of models to pre-train an agent before deploying it in a real-world setting, but modeling the interaction is not always feasible.

RL is most often used to personalize a system's already existing functionality, [10] identifies three main strategies for personalization with RL:

1) **each user is represented by a separate environment** – the goal in such a setup is to identify a set of optimal policies, each tailored to a specific user's preferences and needs. While the difficulty of personalization remains at the level of finding a general policy, the approach requires a large number of iterations and interactions with users, which is often not practical. Another drawback is a lack of knowledge transfer; experience with one user cannot be used to adapt system behavior to the preferences of another user.
2) **learning a single optimal policy with user-specific information** – the approach consists of a single agent learning optimal behavior for each user by providing it with a vector representation describing the user [11]. This can be a set of characteristics relevant to the personalization problem or a unique identifier of the given user. Such a setup overcomes the transfer problem, but introduces a significant growth in state space size. Furthermore, it is often not straightforward which user-specific attributes are relevant.
3) **acquiring personalized behavior for groups of users** – balances the advantages of the previous two approaches by personalizing system behavior not to individual users but a group of users (or environments) based on their similarity [12]. It assumes that experience with similar environments can be useful for personalization. However, defining a clear mapping function that accurately assigns users to the appropriate group is hard; a poorly chosen mapping function can result in a policy that is suboptimal for either of two environments deemed similar by the mapping function, a phenomenon referred to as *negative transfer problem* [13].

Reference [10] provides a systematic overview of research in using RL for personalization. The authors identified an increasing trend of applying RL in this context, with health, entertainment, commerce, and education being the most typical application domains. The survey found that most researchers use value-based RL methods for personalization, Q-learning in particular (99 and 60 papers respectively from a total of 205 papers considered). Another popular approach was the use of multi-armed bandit scenarios (24 papers), with contextual bandits being prevalent within this category (12 papers).

When training multiple agents to personalize to users' needs, multi-agent reinforcement learning (MARL) can be useful to improve the agents' performance. The main advantage of MARL methods is their support for parallelism and a speed-up in training [14], but by keeping the individual agents diverse, it is also possible to support experience sharing.

In MARL, the goal is to reach a better policy by combining the policies of individual agents. The intuition behind this approach is that, through a voting system, the strengths of some algorithms can be enhanced while also mitigating the negative effects of the algorithms' drawbacks [15].

In most MARL setups, agents are trained simultaneously through joint action selections, since without a model it is impossible to simulate the effects of different actions. Ensemble learning, however, uses independently trained RL agents and combines their policies only after training. As it was shown in [16], such an approach can be more efficient with regards to experience, since different agents can explore different policies. Ensemble systems can perform better than multi-agent systems learning from joint action selection.

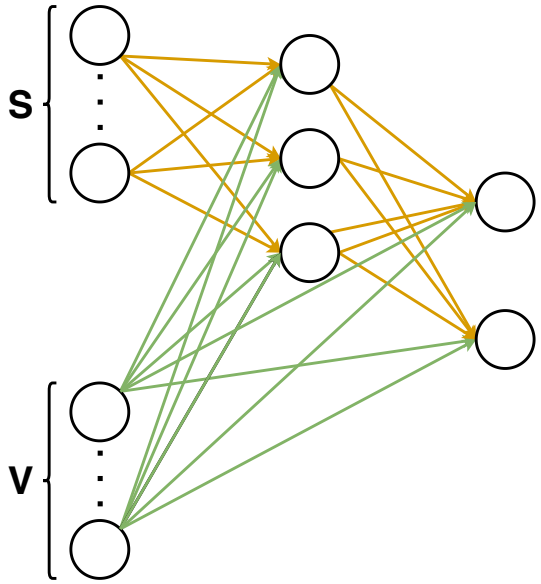

Fig. 1: Dynamically Loaded Biases Q-network architecture

IV. Our Architecture

We propose a novel Q-network architecture tailored to supporting personalization tasks – Dynamically Loaded Biases Q-Network (DLBQN). Our primary goal is to devise an architecture that supports both experience transfer and personalization for environments with continuous state spaces. Another difference between most applications of RL for personalization and DLBQN is that while in the formar cases RL is only used to adjust an already existing system behavior, DLBQN is able to learn a general behavior *and* personalize it at the same time.

Our architecture comprises a single neural network with two types of inputs, as shown in Fig. 1. One input to the network is the state representation (denoted by S), the other is a sparse vector V with a single 1 value identifying the environment the agent is interacting with. The number of input nodes in S is given by the number of parameters describing the environment's state, while the number of input nodes in V is given by the number of environments to which we want to personalize the RL agent's behavior (an environment can correspond to a single user or a group of users).

Topologically, the proposed architecture can be seen as a multi-agent system with the agents sharing most of their parameters. Considering the two input layers, we can divide our architecture into two parts. One part of the network (orange weights in Fig. 1) is responsible for learning a general behavior, since all nodes are directly or indirectly connected to input S. The other half of the network (green weights) depends only on input V, and so is responsible for personalization. Unlike simple Q-network architectures where the vector describing users is part of the state representation (point 2 in section III), in the proposed architecture input V is directly connected to every node in the hidden layers and the output layer of the network. Furthermore, the hidden and output nodes in our architecture have no biases.

If we consider an arbitrary neuron k, the neuron's input function can be calculated according to Equation 3:

$$Z_k = \sum_{j=1}^{m} w_{kj} x_j + b_k, \tag{3}$$

where m is the number of inputs to the neuron, w_{kj} is the value of the synaptic weight corresponding to input j, x_j is an input value to the neuron, and b_k is the neuron's bias.

Since in our architecture each neuron is connected also to input V, and no neuron has a bias, the calculation of the weighted sum is as follows:

$$Z_k = \sum_{j=1}^{m} w_{kj} x_j + \sum_{l=1}^{n} w_{kl} v_l, \tag{4}$$

where n is the size of input V ($|V|$), w_{kl} is the value of the synaptic weight connecting the neuron to the corresponding input value in V, and v_l is an input value from V.

As we know that input V is always a sparse vector with a single 1 value, we can express Equation 4 also as:

$$Z_k = \sum_{j=1}^{m} w_{kj} x_j + w_{kl}, \tag{5}$$

where w_{kl} is the value of the synaptic weight corresponding to v_l if $v_l = 1$. Hence, we can view our architecture as a Q-network with dynamically loaded biases based on the specific environment with which the agent is interacting at a given moment. Personalization therefore takes place in theses biases, while the rest of the weights are responsible for learning a general policy - a policy that works in most cases.

V. Methodology

We tested the personalization ability of our proposed architecture on two simple simulated environments of the gridworld type (see Fig. 2). In this task, the agent must learn to navigate in a 2D environment to reach a goal position by moving in one of four directions: north, east, south, and west. If the agent tries to move beyond the edges of the world, no update of its position takes place. In all our tests, we used a sparse reward function as follows:

$$R = \begin{cases} 1 & \text{if agent\_position=goal\_position} \\ -1 & \text{otherwise.} \end{cases}$$

Our network had three hidden layers with 24, 48, and 24 nodes respectively, we compared its performance and learning ability with a Q-network with an analogous topology. The learning rate was 0.005 for both networks, ε-decay was set to 0.995 with a minimum ε value of 0.01, and the discount factor was 0.85.

We tested the networks first on 4×4 gridworlds (as shown in Fig. 2) with two different state representations. In the first scenario, the state was represented by a tuple of four values as $(x_{agt}, y_{agt}, x_{goal}, y_{goal})$, where agt and $goal$ are the agent and the goal position respectively. In the second setting, the

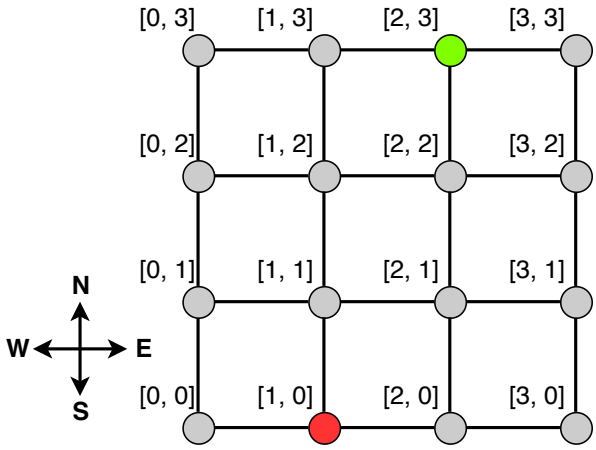

Fig. 2: A 4×4 gridworld environment with the agent at position *[1, 0]* and the goal position at *[2, 3]*

state was represented only by the agent's current position: (x_{agt}, y_{agt}).

To first verify the networks' ability to train with these state representations, we trained them on a single gridworld environment. In further experiments, we trained them on 10 4×4 gridworlds with the goal position generated randomly for each world. In the multi-world scenario, we expected our solution to learn a policy specific for each environment. However, it is important to note that since there are no walls and barriers in the gridworld, in the four-parameter state representation, there is a single optimal policy for all environments: if $x_{agt} < x_{goal}$ then move east, if $y_{agt} < y_{goal}$ then move north, etc.

To better approximate a real-world personalization task, we also tested the two networks on a variant of the gridworld problem – Push the Box. In this scenario, the agent's job is to move a box in cooperation with a simulated human. We used two types of simulated humans: 1) first push the box to the same column as the goal position, then to the same row; 2) first push the box to the same row as the goal position, then to the same column. The state of the environment (position of the box) is updated only when the agent and the human make the same move. In comparison with the test setting of gridworlds, we trained the models on 3×3 environments. Both the goal position and the type of the simulated human cooperator were generated randomly for each environment.

With regards to the environment's behavior, there is no difference between the two test cases, action selection should however be different. For example, in the state depicted in Fig. 2, moving east and north would both be acceptable actions in a simple gridworld. On the other hand, in the Push the Box scenario, the agent would have to consider the simulated human's preferences – with a cooperator of type 1, moving east is the only acceptable action (to get the box to the same column as the goal), while with cooperator 2, the agent must push the box northward.

We trained the networks for 100 iterations per environment, where 1 iteration represented the agent moving in the world until it either reached the goal position or had carried out 100 steps in a gridworld or 50 in the Push the Box scenario. We evaluated the agent's performance by checking whether it would choose an acceptable action in all states except for the goal state. In the gridworld scenario, any action was considered to be acceptable that got the agent closer to the goal position. In Push the Box, the agent's action was accepted if it was the same as the one the simulated human cooperator would carry out in the given state.

In total, we considered 16 test cases: 2 network types $\times$ 2 test environments $\times$ 2 world counts $\times$ 2 state representations. We expected similar functionality from both networks trained on a single world. In multi-world scenarios, we expected our architecture to perform better (except for gridworld with a four-parameter state representation, as discussed earlier).

VI. Results

To evaluate the tested models, we considered the following metrics in all test cases:

1) maximum precision for each test run – Did the agent manage to find an optimal policy?
2) average of maximum precisions for each test run – How good was the average best policy found by the agent?
3) mean precision over the last iterations – How stable was the acquired policy?
4) mean precision over the first iterations – How quickly did the agent converge to the best policy?

Table Ia shows the analysis of the highest precisions accomplished in each test setting. Both models learned an optimal policy on a single world in each test setting, as the maximal precision of 100% was accomplished in every test run. This proves that the topology of the networks is robust enough to support training.

In the multiple gridworld scenario, we expected our architecture to perform better or at an equal level to a simple DQN. While this expectation was met in tests with a two-parameter state representation, DQN still performed better with four-parameter state representation, failing to find the optimal policy only once (the precision was 99.33% in that case, which corresponds to a single faulty action selection from all possible states). It is important to note that the DLBQN architecture acquired a policy containing a single error in 13 tests with four-parameter state representation, and 18 tests with two-parameter state representation. The worst policies had a precision of 88%, and 78%, respectively. The mean best policy was more accurate for DLBQN in two-parameter state representations, with DQN never finding an optimal policy.

For an analysis of the stability of the acquired policies, please refer to Table Ib. A network found a stable policy if there were no significant oscillations in the last iterations of training. When analysing the agents, we considered the mean precision over the last 30 iterations in three intervals: the last 5 iterations, the last 10-5 iterations, and the last 30-10 iterations. When compared with the mean best precisions from Table Ia, the data shows that there is no significant oscillation, except

TABLE I: Results of testing on gridworld environment

agent	state	worlds	max. precision		
			max	count	mean
DQN	2	1	100	100	100
		10	81.3	1	70.48
	4	1	100	100	100
		10	100	99	99.99
DLBQN	2	1	100	100	100
		10	100	67	99.32
	4	1	100	100	100
		10	100	77	99.53

(a) Precision of the best policies acquired in each iteration (%)

agent	state	worlds	mean of last n iterations		
			0-5	5-10	10-30
DQN	2	1	99.99	99.98	99.95
		10	42.43	41.12	41.67
	4	1	99.99	99.93	99.9
		10	98.54	98.45	98.44
DLBQN	2	1	99.99	99.97	99.98
		10	96.43	96.3	96.42
	4	1	99.97	99.93	99.9
		10	96.1	96.1	96.26

(b) Mean precision over the last n iterations (%)

agent	state	worlds	mean of first n iterations		
			0-5	5-10	10-30
DQN	2	1	58.97	83.88	96.99
		10	42.79	52.23	52.04
	4	1	60.61	83.85	97.49
		10	45.93	70.03	87.02
DLBQN	2	1	61.2	82.97	96.68
		10	40.59	50.2	69.12
	4	1	59.25	82.56	96.63
		10	39.3	49.64	70.91

(c) Mean precision over the first n iterations (%)

TABLE II: Results of testing on "Push the Box" environment

agent	state	worlds	max. precision		
			max	count	mean
DQN	2	1	100	100	100
		10	51.25	1	48.75
	4	1	100	100	100
		10	100	8	91.05
DLBQN	2	1	100	100	100
		10	100	96	99.9
	4	1	100	100	100
		10	100	98	99.98

(a) Precision of the best policies acquired in each iteration (%)

agent	state	worlds	mean of last n iterations		
			0-5	5-10	10-30
DQN	2	1	99.98	100	99.99
		10	24.81	24.78	25.41
	4	1	100	99.98	99.99
		10	83.06	83.18	82.98
DLBQN	2	1	100	99.98	100
		10	99.73	99.76	99.72
	4	1	100	100	99.98
		10	99.77	99.77	99.71

(b) Mean precision over the last n iterations (%)

agent	state	worlds	mean of first n iterations		
			0-5	5-10	10-30
DQN	2	1	52.2	85.95	98.21
		10	33.4	33.53	34.14
	4	1	53.38	89.43	98.37
		10	37.43	89.97	75.41
DLBQN	2	1	46.55	84.08	97.81
		10	28.68	41.91	61.38
	4	1	48.03	87.08	98.21
		10	28.71	42.99	63.18

(c) Mean precision over the first n iterations (%)

for the DQN with multiple worlds and two-parameter state representation, which did not converge to a policy at all. This shows that the training was long enough for the agent to learn a stable policy in all other test cases.

Finally, we consider the speed of learning in early phases of training, where a steeper learning curve points at a more efficient learning process. From Table Ic we can see that there is no significant difference in the speed of learning of the two architectures trained on a single world, which is not surprising, since the two models have the same number of parameters and the topologies could be considered equal. In multi-world scenarios, DQN converges faster for four-parameter state representation, and does not converge at all for two-parameter state representation.

The main results of testing our models on the Push the Box environment are shown in Table II. Here we expected similar trends to those from experiments with gridworld environments, but considering the higher need for personalization, we expected the simple DQN model to perform worse.

The test results on a single environment in Table IIa show that the proposed topology is robust enough for optimizing the policy on a single environment. However, in the multi-environment setting we see that DQN performed worse than our architecture. DLBQN performed surprisingly well even in multi-environment settings, failing to find the optimal policy only six times. From these test runs, the model found a policy with a 98.75% precision in four cases, which translates to a single erroneous action selection over all possible states.

Table IIb shows that all models were stable towards the end of training, since we did not observe any significant oscillation in the mean precision of the models' policies over the last training iterations. Similarly to the gridworld test scenario, DQN learned quicker in the early phases of training, as shown by the higher mean precision of the policies in Table IIc. The difference in single-world settings is not significant, while the difference in multi-world tests can be explained by the larger number of parameters in the DLBQN architecture.

VII. Discussion

Section VI presented the results of the analysis of the tested models' performance, in this section, a more detailed discussion of the results and their implication follows.

First we tested the two architectures on a single test environment to check whether the topology is robust enough to enable the agent to find an optimal policy. In all 8 test cases with a single environment, both models successfully learned an optimal policy in all test runs. This shows that when trained on a single world, the DLBQN architecture is fully analogous to DQN, and the weights connected to the input vector V act similarly to the other network's biases.

In multi-world test scenarios, we expected DLBQN to perform at the same level or better than DQN. For both test

environments, DQN was unable to converge to an optimal policy with a two-parameter state representation (x_{agt}, y_{agt}). Therefore, if we were to use DQNs for personalization, we would need to train individual agents for each environment. This results in bad sample efficiency, a lack of transfer of experiences, and a larger number of trainable parameters of the network. If we consider a topology with w weights and b biases, we would need to adjust $n \times (|w| + |b|)$ parameters for n networks. With the DLBQN architecture, however, we only need to add additional biases to support multiple worlds, resulting in a network with $|w| + n \times |b|$ parameters for the analogous use case.

In four-parameter gridworld, DQN was still able to find an optimal policy, and did it more times than DLBQN. However, as we already pointed out in Section V, in this test setup the same policy is applicable in all worlds, and the DQN agent's action selection is dependant only on the current state of the agent, not the world itself. This shows that no personalization was done, in contrast with the DLBQN model that would select a different action in the same state for two worlds even if the goal position was the same. We expect that DLBQN would have been able to increase its precision had we trained it over more iterations.

This, however, shows one of DLBQN's weaknesses in its current proposed architecture, namely a low sample efficiency. DLBQN is unable to generalize its experience over all worlds, which is also supported by the fact that it converges slower than the corresponding DQN (Table Ic). For example, some actions carried out on the edge of the world are invalid and do not result in a change of the agent's state. We would expect DLBQN to generalize this knowledge, e.g. if it learns not to move south when on the lower edge of the world ($y_{agt} = 0$) in one world, it should be able to copy this behavior for every world. By analyzing the final policies of some test runs we found that faulty action selection is often made by choosing an action that is invalid in the given state. This problem could be solved by a more carefully defined reward function, but the generalization ability of DLBQN should nonetheless be addressed in future research.

In tests with the Push the Box environment, which more closely resembles real-life personalization tasks, DLBQN performed clearly better than DQN. Although the difference is not that large in the mean best precision with four-parameter state representation (Table IIa, lines 4 and 8), this is due to smaller variability in the environments. A more telling result in this test setup is that DQN was able to find the optimal policy for all worlds in only 8 test runs, compared to 98 for DLBQN. The higher precision of DLBQN on a slightly smaller environment also suggests that DLBQN would be able to find the optimal policy for gridworlds more often if trained longer.

VIII. Conclusion

In this paper we proposed a novel neural network architecture – Dynamically Loaded Biases Q-Network – to support personalization tasks. Unlike other applications of reinforcement learning in this domain, where RL is only used to adjust an already existing general behavior of the system, DLBQN simultaneously learns the general behavior and personalizes it.

Our preliminary experiments on two test environments show that DLBQN is able to acquire the expected agent behavior and also adjust it based on specific characteristics of a given environment. The policies are specific to the environment, and are different even when we train DLBQN on two completely identical environments.

One weakness that was identified during our experiments was DLBQN's poor ability to generalize experience among environments, which results in small sample efficiency. In future research, we will try to mitigate this problem through a more appropriate training process. We will also try the DLBQN architecture for deep learning, and test it in a real-world personalization task.

Acknowledgment

This research work was supported by APVV project 015-0730 "Cloud Based Human Robot Interaction'.

References

[1] H. Fan and M. S. Poole, "What is personalization? perspectives on the design and implementation of personalization in information systems," *Journal of Organizational Computing and Electronic Commerce*, vol. 16, no. 3-4, pp. 179–202, 2006.

[2] M. A. Hamburg and F. S. Collins, "The path to personalized medicine," *New England Journal of Medicine*, vol. 363, no. 4, pp. 301–304, 2010.

[3] K. N. Martin and I. Arroyo, "Agentx: Using reinforcement learning to improve the effectiveness of intelligent tutoring systems," in *International Conference on Intelligent Tutoring Systems*. Springer, 2004, pp. 564–572.

[4] S. Ferretti, S. Mirri, C. Prandi, and P. Salomoni, "On personalizing web content through reinforcement learning," *Universal Access in the Information Society*, vol. 16, no. 2, pp. 395–410, 2017.

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

[6] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[8] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen, "Collaborative filtering recommender systems," in *The adaptive web*. Springer, 2007, pp. 291–324.

[9] A. Hans, D. Schneegaß, A. M. Schäfer, and S. Udluft, "Safe exploration for reinforcement learning." in *ESANN*, 2008, pp. 143–148.

[10] F. den Hengst, E. M. Grua, A. el Hassouni, and M. Hoogendoorn, "Reinforcement learning for personalization: A systematic literature review," *Data Science*, no. Preprint, pp. 1–41, 2020.

[11] F. Den Hengst, M. Hoogendoorn, F. Van Harmelen, and J. Bosman, "Reinforcement learning for personalized dialogue management," in *IEEE/WIC/ACM International Conference on Web Intelligence*, 2019, pp. 59–67.

[12] A. el Hassouni, M. Hoogendoorn, M. van Otterlo, and E. Barbaro, "Personalization of health interventions using cluster-based reinforcement learning," in *International Conference on Principles and Practice of Multi-Agent Systems*. Springer, 2018, pp. 467–475.

[13] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[14] C. Guestrin, M. Lagoudakis, and R. Parr, "Coordinated reinforcement learning," in *ICML*, vol. 2. Citeseer, 2002, pp. 227–234.

[15] M. A. Wiering and H. Van Hasselt, "Ensemble algorithms in reinforcement learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 38, no. 4, pp. 930–936, 2008.

[16] S. Faußer and F. Schwenker, "Ensemble methods for reinforcement learning with function approximation," in *International Workshop on Multiple Classifier Systems*. Springer, 2011, pp. 56–65.