

Complementary reinforcement learning toward explainable agents

Jung Hoon Lee¹

¹Allen Institute for Brain Science

Abstract

Reinforcement learning (RL) algorithms allow agents to learn skills and strategies to perform complex tasks without detailed instructions or expensive labelled training examples. That is, RL agents can learn, as we learn. Given the importance of learning in our intelligence, RL has been thought to be one of key components to general artificial intelligence, and recent breakthroughs in deep reinforcement learning suggest that neural networks (NN) are natural platforms for RL agents. However, despite the efficiency and versatility of NN-based RL agents, their decision-making remains incomprehensible, reducing their utilities. To deploy RL into a wider range of applications, it is imperative to develop explainable NN-based RL agents. Here, we propose a method to derive a secondary comprehensible agent from a NN-based RL agent, whose decision-makings are based on simple rules. Our empirical evaluation of this secondary agent’s performance supports the possibility of building a comprehensible and transparent agent using a NN-based RL agent.

Introduction

Reinforcement learning (RL), inspired by our brain’s reward-based learning, allows artificial agents to learn a wide range of tasks without detailed instructions or labeled training sets which are necessary for supervised learning (Hertz et al., 1991; Sutton and Barto, 1998). Given that RL agents’ learning resembles our process of learning and that learning is essential to our intelligence, it seems natural to assume that RL is one of the key components to brain-like intelligent agents or general artificial intelligence. Recent breakthroughs (Mnih et al., 2015; Silver et al., 2016) from DeepMind team showed that RL could train neural network (NN)-based agents to outperform humans in video-games and even ‘Go’, reigniting interests in RL and its applications in NN-based agents, and noticeable developments in NN-based RL agents have been reported since then; see (Mirowski et al., 2017; Mnih et al., 2016; Oh et al., 2016; Plappert et al., 2018; Wang et al., 2016) for examples.

However, despite rapid improvements in RL, NN-based RL agents’ decision-making process remains incomprehensible to us. For instance, the AlphaGo’s strategies employed during the match with Sedol Lee exhibited efficiency leading to victories, but the exact reasons behind its moves are unknown. AlphaGo demonstrated that incomprehensible decision-making can still be effective, but its effectiveness does not mean that it cannot be faulty. In some applications, even sporadic mistakes must be prevented. A loss in one Go match out of 100 matches is insignificant, but crashing a car once out of 100 drives is perilous and unacceptable. To ensure that NN-based RL agents work properly, predict their failures and fix their mistakes if necessary, it is imperative to understand the principles underlying their decision-making. For brevity, NN-based RL agents will be referred to as RL agents throughout this study.

Two earlier studies (Hayes and Shah, 2017; Waa et al., 2017) showed that the decision-making processes of RL agents can be translated into human-readable descriptions. These proposed algorithms do provide some insights into decision-making process of RL agents, but they do not address how we could modify RL agents’ actions because their study focused on providing interpretation of RL agents’ action. To fully understand internal mechanisms of RL agents, we need to build an agent that relies on symbolic rules to make decisions. However, developing symbolic rules for specific domains has been proved challenging, and thus it will be extremely difficult to find a golden set of rules to be applied to general RL problems. Then, how do we build explainable RL agents? We propose a secondary agent as a potential solution, which utilizes simple rules to choose the best action. This proposal is based on two ideas. First, the secondary agent’s action can be analyzed because it utilizes simple rules. Second, the secondary agent can perform general tasks, if it takes advantage of the trained RL agents.

In this study, we propose a quasi-symbolic (QS) agent as a secondary agent and compare its performance to RL agents’ performance. Specifically, QS agents learn values of transitions of states from RL agents’ behaviors. After learning the values of transitions, QS agents identify the most valuable states (hub states) and search for a sequence of actions (action plans) to reach one of the hub states. If QS agent cannot find a proper action plan to reach a hub state, they choose the best action with the values of transition. The QS agents are based on generic algorithms to examine the plausibility of constructing secondary agents complementary to RL agents. In this study, we tested QS agents’ performance using the ‘lunar-lander’ benchmark problem available in open-AI gym environments (Brockman et al., 2016). Our experiments show that QS agents’ performance is comparable to RL agents’ performance. While our experiments are conducted in a simple environment, the results indicate that comprehensible agents with transparent decision-making process can be derived from RL agents.

Methods

QS agents interact with RL agents and make plans for future actions by utilizing the model of environment (Env network). All these three networks/agents are constructed using open-source machine learning toolkit (Paszke et al., 2017). Below, we discuss QS, RL and Env network in details.

The structure of reference RL agent

We used an actor-critic model as a reference RL agent (Grondman et al., 2012; Konda and Tsitsiklis, 2000). The implementation was adopted from an official pytorch github repository (Pytorch-team, 2018). The actor consists of 8 input, 100 hidden and 4 output nodes, and the critic, 8 input, 100 hidden and 1 output nodes. The initial learning rate is 0.003 and is decreased by 10 times at every 1000 episode.

The structure of quasi-symbolic (QS) agent

QS agent consists of matching and value networks, which are both single (synaptic) layer networks. The matching and value networks are sequentially connected (Fig. 1A). That is, the output node of matching network is directly connected to the input node of value network. The number of matching nodes (output nodes in the matching network) is identical to the number of value nodes (input nodes in the value network), and the connections between matching and value networks

are one-to-one. The matching network memorizes input vectors by imprinting normalized inputs to synaptic weights converging onto the output node O_i , as shown in Eq. 1; h_i the synaptic input to O_i .

$$h_i = \sum_j w_{ij} \frac{s_j^k}{\|s_k\|}, \text{ where } w_{ij} = \frac{s_i^l}{\|s_l\|} \quad (1)$$

, where indices i and j represent components of state vectors, and indices k and l represent state vectors. With normalized inputs stored in synaptic weights w_{ij} , the outputs of matching network represent cosine similarities between the current input and stored inputs. If all synaptic inputs h_i to all output nodes O_i are smaller than the threshold value θ , the input is considered novel and stored into the matching network by adding a new output node into the matching network; the default threshold is $\theta=0.97$, unless stated otherwise. Once a new node is added to the matching network, a new node is also added to the value network, to keep one-to-one mapping (Fig. 1A). The connection between these two newly established units is determined by the reward associated with the input vector. When the current input is one of the previously stored ones (i.e., one of synaptic input to output nodes is higher than the threshold θ), the maximally activated node is identified, and the connection to the corresponding value network node is updated by adding the reward induced by the input vector (Eq. 2).

$$W_k \leftarrow W_k + r_t \quad (2)$$

, where W_k is the connection between the matching node m_k and value node v_k , and r_t is the reward the RL agent obtain at time t .

In this study, the matching network is designed to memorize the transitions between states S and S' . That is, the input to the network is $\Delta S = S' - S$. Since there are 8 state variables in the lunar-lander benchmark problem, the number of input nodes of the matching network is 8. The sizes of the matching and value networks are not fixed. Instead, they are determined during the training of QS agents. The more diverse inputs are, the bigger matching networks become. For instance, if all inputs are identical, there is only one matching node.

Env network

QS agents utilize the Env network to model the environment; it means that QS agent can be considered model-based agents. As this network models the environment, it receives the state vector and action as inputs and returns the next state (Fig. 1B). The inputs layer includes 12 nodes which represent 8 state variables and 4 possible actions in the lunar lander environment, and the output layers include 8 nodes (due to 8 state variables). In our experiments, we set the hidden layer of the Env network to have 300 nodes. The Env network is trained using the mean squared error (squared L2 norm). In each episode of RL training, the error is accumulated. After the episode, the Env network is updated using the accumulated error; that is, 1 episode is a single batch of backpropagation. The initial learning rate is 0.05 and is decreased to 10 percent at every 1000 episode.

Results

In this section, we discuss the operating principles of QS agents including the interplay between QS and RL agents. Then, we present our experiments which were conducted to evaluate QS agents' performance in solving lunar-lander problem compared to RL agents.

Training and operating rules of Quasi-Symbolic (QS) agents

Unlike RL agents, QS agents do not work alone. Instead, their operations depend on RL agents in learning and inference modes; that is, RL and QS agents are complementary with each other. The matching and value networks in QS agents are updated by using RL agents' behaviors during training. Specifically, in each time step in the training period, the previous state S and the current state S' are used to generate the state transition vectors $\Delta S = S' - S$, and this transition vector is fed into the matching network as inputs. The matching network first determines the novelty of the current transition vector; this novelty detection is done by inspecting synaptic inputs to matching nodes (see Methods for details). When a novel transition vector is introduced, a new output node is added to the matching and value networks (Fig. 1A), and the connection between matching and value networks (one-to-one connection w_k between newly added nodes, m_k and v_k) is established (Methods). The strength of the connection w_k is determined by the reward given to the RL agent after the transition from S to S' . When the earlier input is introduced again, the maximally activated matching node is identified, and the connection between the identified matching node and the value node, which is exclusive and one-to-one (Fig. 1A), is updated by adding the reward obtained to the previous strength (Eq. 2). In brief, the matching network memorizes transition vectors observed during training, and the value network stores the amount of rewards induced by the observed transitions.

In the inference mode, in which QS agents choose the best action, QS agents utilize both the trained RL agents and Env networks to make action plans, whose lengths are variable. In doing so, QS agents identify the most valuable transition vectors ΔS_k by inspecting connections' strength between matching and value networks. In this study, we identified synaptic weights higher than θ_{hub} defined in Eq. 3.

$$\theta_{hub} = m(W_k) + \alpha \cdot s(W_k) \quad (3)$$

, where m and s are mean and standard deviations of synaptic weights, and α is the scale constant which determines the magnitude of θ_{hub} . The default value of α is 0.1. Due to the one-to-one connections between matching and value networks, the synaptic weights allow us to identify the most valuable transitions observed during the training period, which are referred to as 'hub' states hereafter.

After identifying the hub states, QS agents search for an action plan to reach one of the hub states. In doing so, QS agents utilize the Env network and the trained RL agent to predict future states (and thus transitions as well) to make an action plan. Given the state, RL agent provides a possible action, and Env network returns the next state. Employing them recursively (Fig. 1B), QS agents can predict the future states. During this planning, at each time step, QS agents examine whether a transition vector is one of the hub states (i.e., the most valuable transitions) or not. If QS agents do expect to reach one of the hub states, they stop planning and executes the current plan. The maximum length of the action plan is 10 time-step, unless stated otherwise. If no hub state cannot

be reached within the predefined maximal time-step, QS agents start over and make a new plan. For each state, QS agents are allowed to make a total of 5 different plans. If they cannot find a path to the hub states in all five plans, they inspect the values of the first transition in the five plans and choose the best immediate action to produce the best transition using the value network. In this case, rather than taking a sequence of actions, QS agents execute the first action only.

Next, to evaluate QS agents’ performance, we compared QS agents’ performance to that of RL agents by using the “lunar-lander” benchmark task included in the OpenAI gym environment (Brockman et al., 2016).

QS agents’ performance compared to RL agents

In this study, we constructed RL agents (actor-critic model), QS agents and Env network using ‘pytorch’, an open-source machine learning library (Paszke et al., 2017); their schematics are illustrated in Fig. 1. We trained RL, QS agents and Env network during 5000 episodes; see Methods for training details. Figures 2A and B show the total amount of rewards given to the RL agent and the error function of the Env network during 5000 episodes. As shown in the figure, while the reward in each trial fluctuates from one trial to another, the amount of rewards, on average, increases rapidly until 1000 episodes. After 1000 episodes, the speed of improvement is reduced. Similarly, the error of Env network is reduced most rapidly in the first 1000 episodes, and then the speed of error-reduction slows down.

At every 1000 episode, we froze the learning and tested both QS and RL agents using the same 100 environments of lunar lander; that is, the environments are instantiated with the same random seeds. The RL agent in this study uses stochastic policy to choose actions, and thus its behaviors depend on the random seed forwarded to the pytorch. Moreover, QS agents’ behavior are also stochastic, as they rely on RL agents for their decisions. To avoid potential biases based on their stochastic behaviors, we constructed 10 independent QS and RL agents by forwarding distinct random seeds to pytorch and calculated the average reward for both agents. Figures 3A and B show the average amount of QS and RL agents for all 100 instantiations of environment. x -axis represents the identity of environment, and y -axis represents the reward averaged over 10 independently constructed agents. Rewards are estimated after training the RL agent in 1000 episodes (Fig. 3A) and 5000 episodes (Fig. 3B). As shown in the figure, QS agent with 10 time-step action plan shows slightly worse performance (shown in red) than RL agents’ performance (shown in blue). Figure 3C shows the average reward calculated using 10 independent agents in 100 environments, after training the RL agent in 1000, 2000, 3000, 4000 and 5000 episodes, respectively; that is, the mean values and standard errors are estimated from 1000 individual experiments.

Next, we varied the parameters to examine how they affect QS agents’ performance. First, we tested the effects of action plans’ lengths. As shown in Fig. 4A, QS agents’ performance improves, as they generate longer action plans. Second, we increased the number of hub states by lowering θ_{hub} . Specifically, we set $\alpha=0.05$ and estimated the amount of rewards. As shown in Fig. 4B, QS agents’ performance improved. We also increased θ_{hub} to further examine the effects of the number of hub states on QS agents’ performance and found that QS performance is negatively correlated with θ_{hub} (Fig. 5A). Finally, we perturbed the threshold value θ for the novel detection (used in matching networks) to see how it affects QS agents’ performance. As the threshold θ becomes

lower, the accuracy of predictions on QS agents' future states decreases. As shown in Fig. 5B, QS agents obtained less rewards.

Discussion

In this study, we propose a method to derive a secondary agent from RL agents for more transparency. The newly proposed QS agents have two operating units. The matching network memorizes the state-transition ΔS , and the value network associates each state-transition to the reward RL agents obtained. In this way, QS agents can evaluate probable actions suggested by RL agents. Importantly, QS agents do not just evaluate the values of immediate actions but also plan future actions. As shown in Fig. 4A, QS agents' performance improves when they establish longer plans, suggesting that QS agents' ability to plan ensures their utilities. More importantly, QS agents utilize simple rules to make decisions, and a single matching node is active one at a time. Due to these properties, their internal mechanisms can be easily understood, which facilitates modification when necessary.

Potential variation of QS agents

QS agents are built based on the assumption that a transition from the current state S to the future state S' is sufficient for evaluating RL agents' behaviors. In fact, this assumption is consistent with the Markov decision process (MDP) framework, in which the transitions between the states are independent from the history of states that are earlier than the current state. It suggests, in principle, that current QS agents can be used for any problems that can be formulated as MDP. However, it does not mean that QS agents perform well with other RL problems. The algorithm we propose here is designed to be as generic as possible. Instead of refining the algorithm, we focus on addressing the possibility of deriving a comprehensible secondary agent from RL agents. We expect that some variations of QS agents would be necessary for other RL problems. Below, we list a few potential variations of our generic QS agents.

First, current QS agents evaluate individual transitions ΔS using the rewards RL agents obtain immediately after completing the transition. However, the actual values of transitions can be estimated differently. For instance, instead of using the immediately obtained reward, we can use the total rewards from the transition to the end of episodes for evaluating state transitions.

Second, the state-transition vectors can be split into multiple vectors so that value vectors, rather than a single scalar value, can be generated for transitions. This split could be useful for some problems, since state variables do not have equivalent values in agents' behaviors. For example, the coordinates and velocities included in the state vectors of lunar-lander do have different importance in dynamics of agents. If we deal with coordinates and velocities distinctively, QS agents may evaluate the state transitions more effectively. This idea can be implemented by using multiple matching and value networks in QS agents and calculating a linear sum of value networks. Ideally, the linear sum needs to be adaptively updated using learning algorithms.

Third, the state transition vectors can be coupled with state vectors to estimate the values of actions more precisely. For instance, an agent's horizontal move can be either bad or good depending on the current state. If both state and transition vectors are used, QS agent may have better estimation of agents' actions.

Implications for the brain's complementary system

Prefrontal cortex (PFC) has been thought to be a hub for high-level cognitive functions such as decision-making, learning and memory (Lara and Wallis, 2015; Miller and Cohen, 2001; Tanji and Hoshi, 2008; Wang, 2012). However, PFC does not work alone and is known to be connected to other brain areas. The two main areas that have strong interactions with PFC are hippocampus (Jin and Maren, 2015; Peyrache et al., 2011) and anterior cingulate (ACC) (Gehring and Knight, 2000; Paus, 2001). Interestingly, complementary learning system theory suggests central roles of the interplay between PFC and hippocampus in our ability to learn continuously (O'Reilly et al., 2014). Then, why does PFC need to interact with ACC? ACC has been postulated to be associated with multiple functions (Vassena et al., 2017) such as error likelihood (Brown and Braver, 2005) and prediction of expected outcomes (Vassena et al., 2014). We note that QS agents can predict the future outcomes as ACC does. Based on our result that QS agents can evaluate RL agents' decisions, we propose that one of ACC functions is to supervise PFC to make the best decision depending on the context.

Acknowledgement

JHL wishes to thank the Allen Institute founders, Paul G. Allen and Jody Allen, for their vision, encouragement and support.

References

- Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). OpenAI Gym. ArXiv.
- Brown, J.W., and Braver, T.S. (2005). Learned Predictions of Error Likelihood in the Anterior Cingulate Cortex. *Science* (80-.). 307, 1118–1121.
- Gehring, W.J., and Knight, R.T. (2000). Prefrontal-cingulate interactions in action monitoring. *Nat. Neurosci.* 3, 516–520.
- Grondman, I., Busoniu, L.B., Lopes, G.A.D., and Babuška, R. (2012). A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients. *IEEE Trans. Syst. MAN Cybern.* 42, 1291–1307.
- Hayes, B., and Shah, J.A. (2017). Improving Robot Controller Interpretability and Transparency Through Autonomous Policy Explanation. In *Human-Robot Interaction (HRI2017)*, p.
- Hertz, J., Krogh, A., and Palmer, R. (1991). *Introduction to the theory of neural computation* (Westview).
- Jin, J., and Maren, S. (2015). Prefrontal-Hippocampal Interactions in Memory and Emotion. *Front. Syst. Neurosci.* 9, 1–8.
- Konda, V., and Tsitsiklis, J.N. (2000). Actor-Critic Algorithms. In *NIPS*, pp. 1008–1013.
- Lara, A.H., and Wallis, J.D. (2015). The Role of Prefrontal Cortex in Working Memory: A Mini Review. *Front. Syst. Neurosci.* 9, 1–7.
- Miller, E.K., and Cohen, J.D. (2001). An Integrative Theory of Prefrontal Cortex Function. 167–

- Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A.J., Banino, A., Denil, M., Goroshin, R., Sifre, L., Kavukcuoglu, K., et al. (2017). LEARNING TO NAVIGATE IN COMPLEX ENVIRONMENTS Piotr. In ICLR, p.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*.
- Mnih, V., Mirza, M., Graves, A., Harley, T., Lillicrap, T.P., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. In International Conference on Machine Learning, p.
- O'Reilly, R.C., Bhattacharyya, R., Howard, M.D., and Ketz, N. (2014). Complementary learning systems. *Cogn. Sci.* 38, 1229–1248.
- Oh, J., Chockalingam, V., Singh, S., and Lee, H. (2016). Control of Memory, Active Perception, and Action in Minecraft. In International Conference on Machine Learning, p.
- Paszke, A., Gross, S., Chintala, S., Chanan, Gregory Yang, E., DeVito, Z., Lin, Zeming Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in PyTorch. In NIPS-W, p.
- Paus, T. (2001). Primate anterior cingulate cortex: where motor control, drive and cognition interface. *Nat. Rev. Neurosci.*
- Peyrache, a., Battaglia, F.P., and Destexhe, a. (2011). Inhibition recruitment in prefrontal cortex during sleep spindles and gating of hippocampal inputs. *Proc. Natl. Acad. Sci.* 108, 17207–17212.
- Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., et al. (2018). Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research. *ArXiv* 1–16.
- Pytorch-team (2018). Pytorch reinforcement examples.
- Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 484–489.
- Sutton, R.S., and Barto, A.G. (1998). No Title (Cambridge, MA: MIT Press).
- Tanji, J., and Hoshi, E. (2008). Role of the lateral prefrontal cortex in executive behavioral control. *Physiol. Rev.* 88, 37–57.
- Vassena, E., Silvetti, M., Boehler, C.N., Achten, E., Fias, W., and Verguts, T. (2014). Overlapping neural systems represent cognitive effort and reward anticipation. *PLoS One* 9.
- Vassena, E., Holroyd, C.B., and Alexander, W.H. (2017). Computational models of anterior cingulate cortex: At the crossroads between prediction and effort. *Front. Neurosci.* 11, 1–9.
- Waa, J. Van Der, Diggelen, J. Van, and Neerincx, M. (2017). Contrastive explanations for

reinforcement learning in terms of expected consequences.

Wang, X.J. (2012). Neural dynamics and circuit mechanisms of decision-making. *Curr. Opin. Neurobiol.* 1–8.

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., and de Freitas, N. (2016). Dueling Network Architectures for Deep Reinforcement Learning. *ArXiv*.

Figures

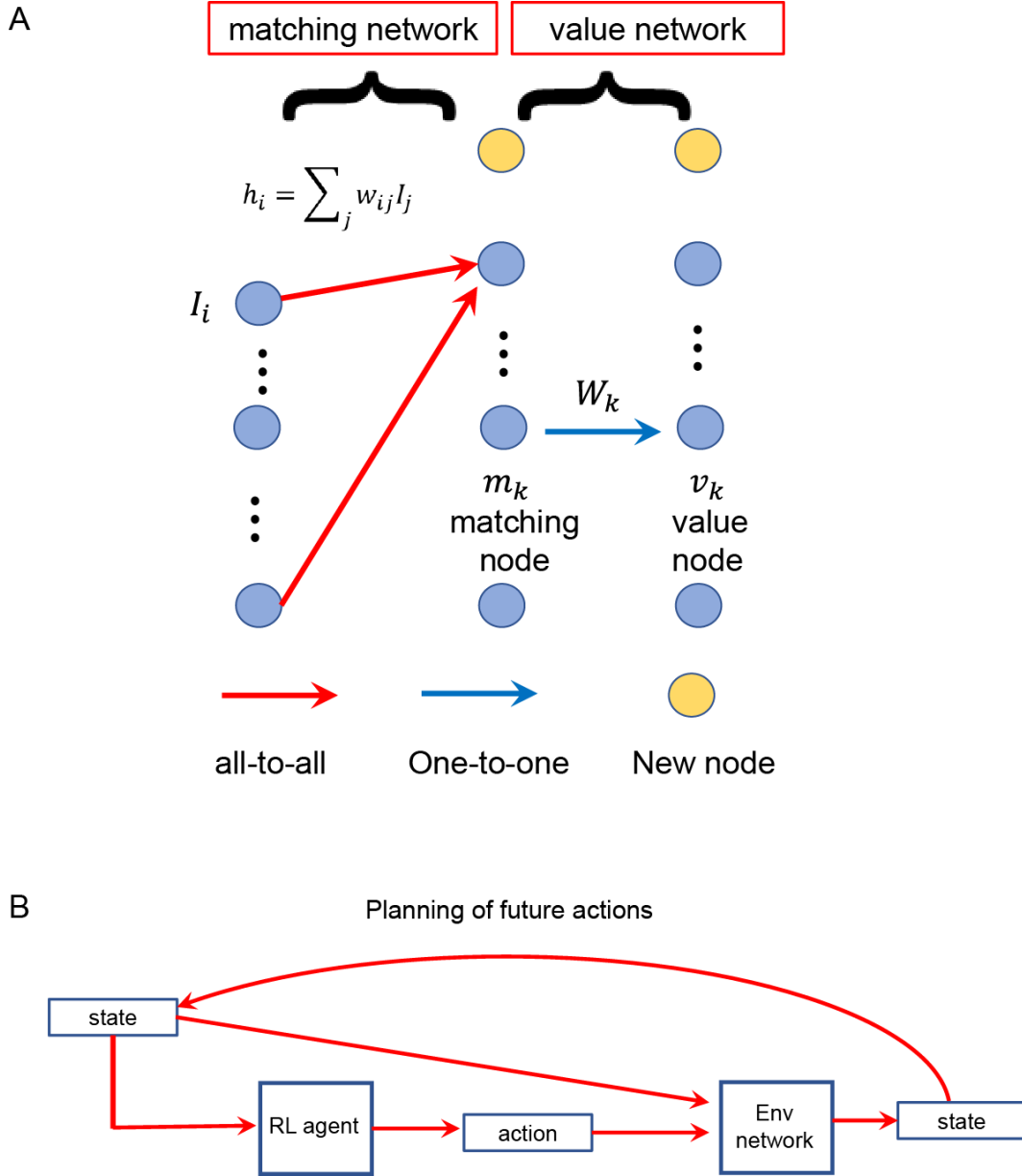


Figure 1: The schematics of models. (A), the structure of the QS agents consisting of matching and value networks. The matching nodes (i.e., the outputs of matching network) are connected to the value nodes via one-to-one connection. When the input vector is novel (Methods), new nodes are added to the matching and value networks and are connected via exclusive one-to-one connection. That is, the number of value nodes is the same as that of matching nodes. (B), the planning of future actions. At each time step, a QS agent utilizes a RL agent for a probable action and feeds the suggested action into Env network to predict the next state. By recursively using them, QS agents can make action plans.

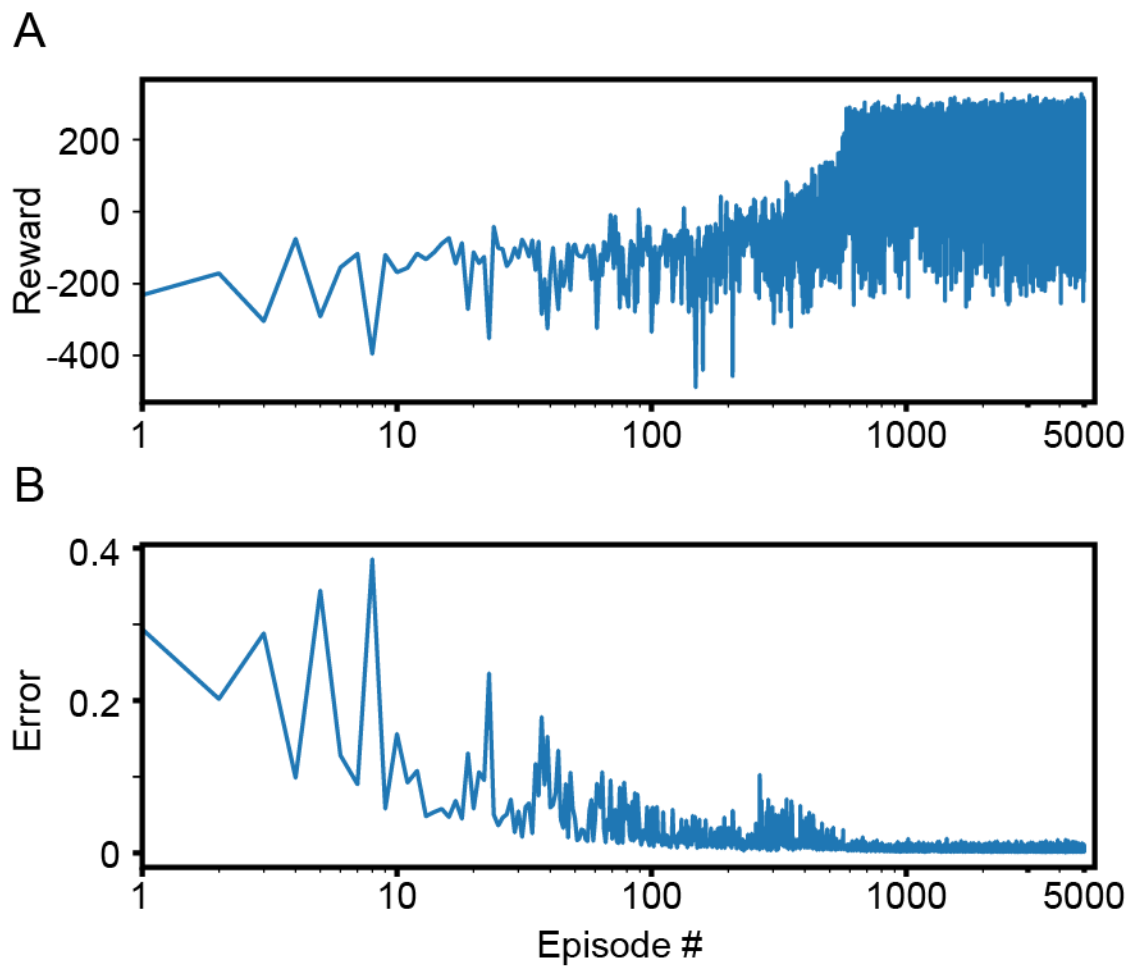


Figure 2: Training of RL agent and Env network. (A), the total reward in each episode. (B), the error observed during training of the Env network.

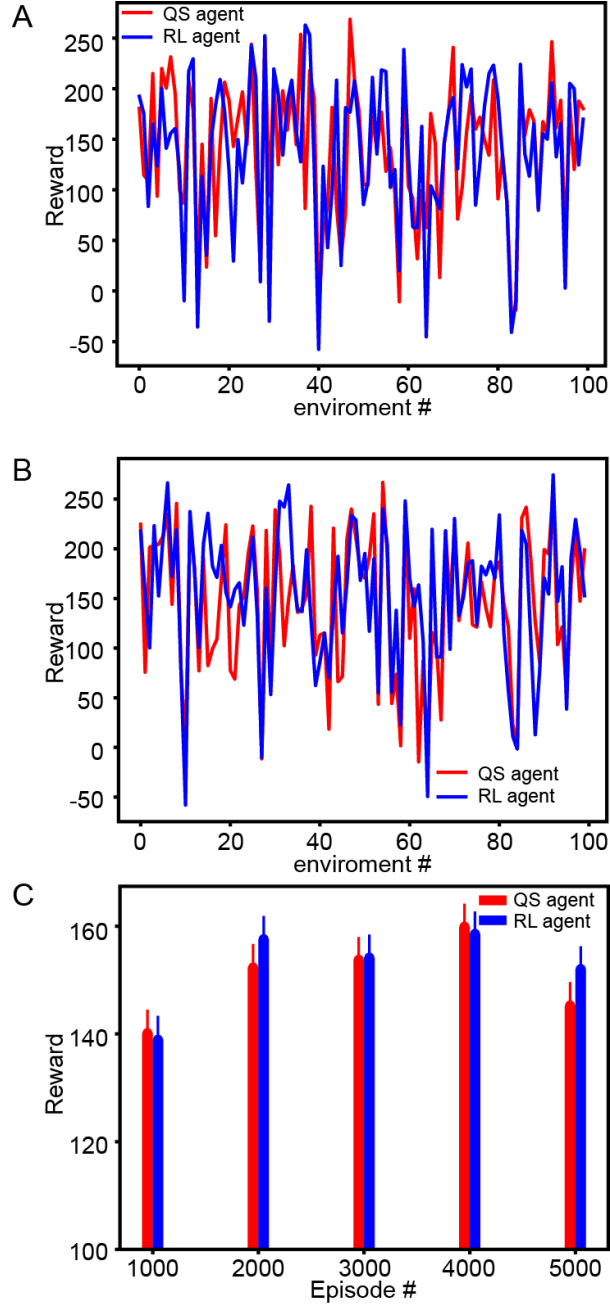


Figure 3: Comparison between QS and RL agents' performances. (A), the average reward of QS and RL agents in 100 environments, after the RL agent is trained in 1000 episodes. 10 independent QS and RL agents are tested for the same environment, and the rewards averaged over 10 agents are shown. The red and blue lines represent QS and RL agents, respectively. (B), the same as (A), but the RL agent is trained during 5000 episodes. (C), the average reward of QS and RL agents depending on the length of RL agents training. The mean values of the rewards are estimated using 10 agents tested in 100 environments. That is, the mean values illustrated are rewards averaged over 1000 individual experiments. The error bars are standard errors from the same 1000 experiments.

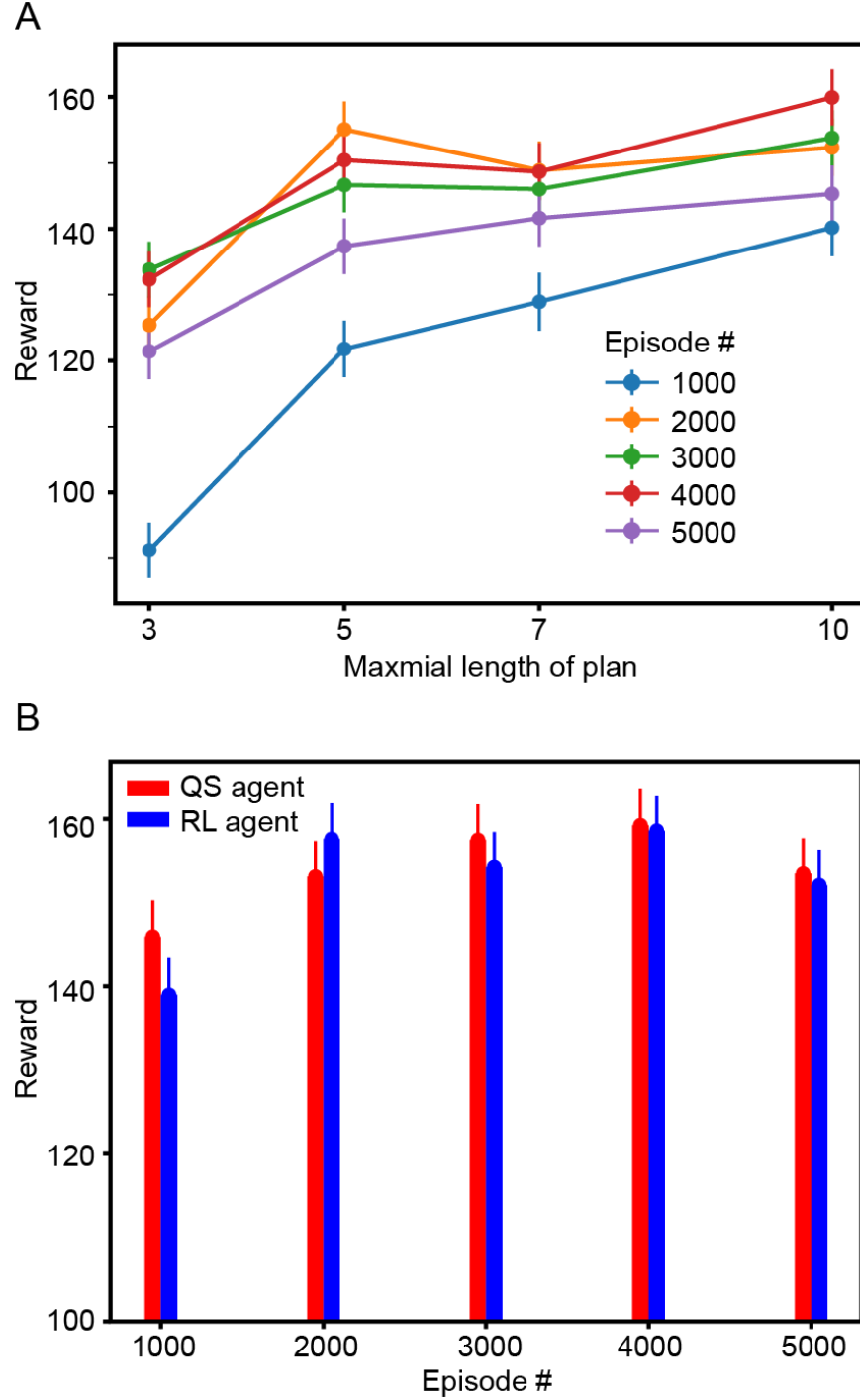


Figure 4: The effect of parameters on the QS agents' performance. (A), the average rewards depending on the maximal length of action plans. The mean values and standard errors of the rewards are estimated using 10 agents tested in 100 environments. We estimated them with RL agents trained during 1000, 2000, 3000, 4000 and 5000 episodes. The color codes represent the length of RL agents training. (B), the average rewards with a bigger set of hub states. The threshold value for the hub states is lowered to increase the number of states.

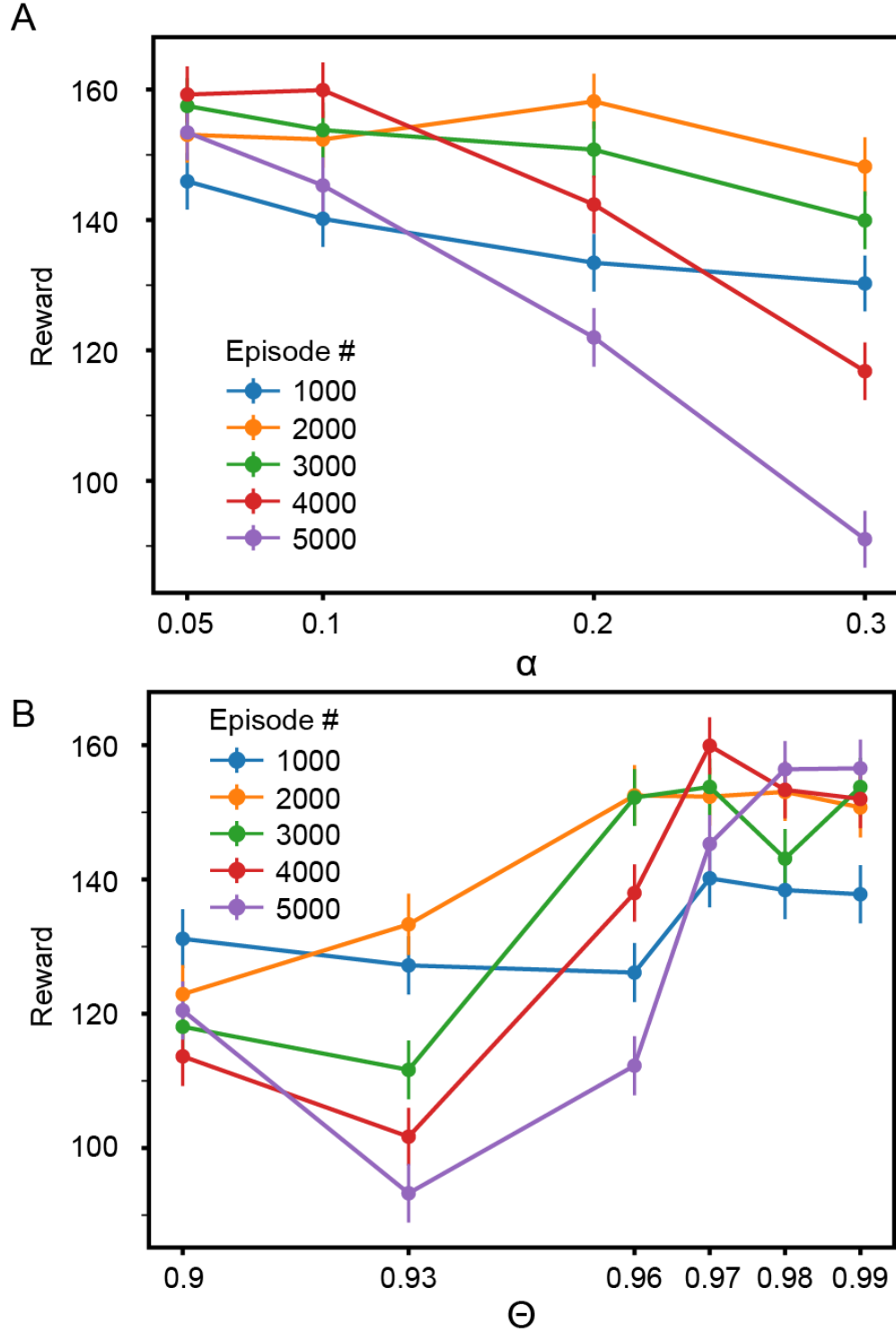


Figure 5: The effect of parameters on the QS agents' performance. (A), the average rewards depending on the threshold value for the hub states (see Eq. 3). **(B),** the average rewards depending on the threshold value θ for novel input detection (Methods).