Emergence of Flexible Prediction-Based Discrete Decision Making and Continuous Motion Generation through Actor-Q-Learning

Katsunari Shibata
Dept. of Electrical and Electronic Engineering
Oita University, Oita, JAPAN
Email: shibata@oita-u.ac.jp

Kenta Goto
Dept. of Electrical & Electronic Engineering
Oita University, Oita, JAPAN
Presently Nabtesco Corporation, JAPAN

***Abstract*—In this paper, the authors first point the importance of three factors for filling the gap between humans and robots in the flexibility in the real world. Those are (1)parallel processing, (2)emergence through learning and solving "what" problems, and (3)abstraction and generalization on the abstract space. To explore the possibility of human-like flexibility in robots, a prediction-required task in which an agent (robot) gets a reward by capturing a moving target that sometimes becomes invisible was learned by reinforcement learning using a recurrent neural network. Even though the agent did not know in advance that "prediction is required" or "what information should be predicted", appropriate discrete decision making, in which 'capture' or 'move' was chosen, and also continuous motion generation in two-dimensional space, could be acquired. Furthermore, in this task, the target sometimes changed its moving direction randomly when it became visible again from invisible state. Then the agent could change its moving direction promptly and appropriately without introducing any special architecture or technique. Such emergent property is what general parallel processing systems such as Subsumption architecture do not have, and the authors believe it is a key to solve the "Frame Problem" fundamentally.**

I. Introduction

It is a great ability for humans to behave flexibly and appropriately in the real world filled with a huge amount of information, and the difficulty in achieving it in robots has been standing as a great barrier that blocks them from being intelligent. The authors believe that there are three main factors required to achieve the ability. One of them is a parallel processing that enables us to consider many things in parallel and to make appropriate decisions immediately without being disturbed by the "Frame Problem" [1]. However, it is very difficult to design the parallel processing because our consciousness is not parallel but sequential. "Subsumption Architecture" [2] is a kind of parallel system, but the interface design among the modules (layers) must disturb its flexibility. Acquiring non-preinstalled ability through experiences is also expected. Accordingly, learning with high degree of freedom in a parallel processing system is essential.

When we develop a function in a robot through learning, we usually focus on its algorithm, in other words, "how a function, for example, recognition of a ball, is realized", but it is really important to solve "What" problems, such as "what function is required" or "what pattern should be recognized" before solving the "How" problems. Brooks mentioned a similar thing to abstraction process [2]. However, it is seriously time-consuming to find the solution to the "What" problems by examining almost infinitive number of possible candidates one by one. Thus, another key factor from this viewpoint is emergence of necessary functions and internal representations through autonomous learning in a parallel system without any directions from outside.

We, in the future, will not meet completely the same sensor signals as present probably. However, in humans, what was learned in a situation is utilized sophisticatedly in a similar situation for its purpose even though the sensor signals are different. Accordingly, the final factor is abstraction from sensor-signal space and generalization on the abstract space.

Therefore, to raise an intelligent robot to a new stage, it seems promising to let a parallel processing-and-learning system learn autonomously as our brain is doing. What meets the condition is the combination of a neural network (NN) and reinforcement learning (RL) [3]. To deal with memory or dynamics in the NN, a recurrent structure is essential to be introduced. The combination of the recurrent neural network (RNN) and RL has been applied to memory-required or dynamical tasks, but it is limited in simple tasks such as "pole-balancing" problems [4]–[6] and "maze with discrete action" problems [7], [8] due to the difficulty in learning. Recently, our group has shown its usefulness in communication [9], context-based behavior [10], exploration [11] and retrospective understanding of pattern meaning [12].

On the other hand, it has been considered useful for developing an intelligent robot or agent that an RNN learns to predict future sensor signals from the present sensor signals and action, and the intermediate representation of the RNN is used as an abstract and context-considered state representation [4], [13], [14]. However, when the number of sensor signals is huge, all of them cannot be predicted and prediction targets have to be chosen among them by a designer. That means a "What" problem mentioned above has come out. The authors showed [15] that by Q-learning [16] with an RNN, an agent (robot) could learn to capture a moving object in a two-dimensional space where it often became invisible. The agent learned a prediction-required series of actions only from rewards and punishments without giving "prediction is necessary" or "what is the prediction target" explicitly.

In this paper, a similar "invisible-target-capture" task is employed, but it is far more difficult in many points such that

two-dimensional continuous motion should be learned instead of one-dimensional discrete action as in [15]. Furthermore, while the target is invisible, it sometimes changes its moving direction and then appears again. Therefore, when the target is invisible, the agent has to predict the target location from the signals while the target was in sight, but if the target appears again and its motion contradicts to the predicted one, the agent has to switch its motion flexibly to the one based on the new prediction.

In this task, two-dimensional continuous motion has to be generated, and also a discrete decision of 'capture' or 'move' should be made. To learn both of them simultaneously, Actor-Q-learning [17] is employed. Actor-Q-learning has not been applied yet to any tasks except for an active perception-and-recognition task [17]. In this paper, the effectiveness of Actor-Q-learning in another task is examined and whether it works with an RNN is also examined. Furthermore, a network structure and initial connection weights of the RNN are also tested.

II. Reinforcement Learning (RL) with a Neural Network (NN) [3]

Reinforcement learning (RL) is autonomous and purposive learning based on trials and errors, and a neural network (NN) is usually used as a non-linear function approximator to avoid the state explosion due to the curse of dimensionality. As mentioned above, our group has claimed that the combination of them enables parallel consideration, function emergence, abstraction and generalization. Through learning, necessary functions such as recognition and memory (when using RNN) emerge only from rewards and punishments to get rewards and/or to avoid punishments. The flexible and parallel processing is expected to contribute to solving the "Frame Problem" [1] or the "What" problems by saying goodbye to the "Functional Modules" approach, in which each functional module is sophisticatedly programed independently and such modules are integrated to develop an intelligent agent.

A. Actor-Q-learning with a Recurrent Neural Network (RNN)

Actor-Q-learning is a learning method for the simultaneous learning of both discrete decision making, which is called action here, and continuous motion generation [17]. Actor-Critic [18] is combined with Q-learning [16], and one of the Q-values is used on behalf of the Critic. Here, a recurrent neural network (RNN) is used for the learning. The system is consisted of one RNN whose inputs are sensor signals. In Actor-Q-learning, the outputs are divided into two kinds: Q outputs and Actor outputs. The Q outputs are responsible to making a discrete decision or action selection. A decision is made according to the Q-values as in the case of the regular Q-learning. The Actor outputs are responsible to continuous motion generation. Actual motion is chosen stochastically in the range whose center is decided by the Actor outputs. That is, the actual motion is the sum of the Actor output vector $\mathbf{m}(\mathbf{s}_t)$ and a random number vector $\mathbf{rnd}_t$ for exploration where $\mathbf{s}_t$ is the input (sensor signal) vector at time t.

As for the training, Q-outputs are trained according to the Q-learning algorithm, and Actor outputs are trained in the same way as Actor-Critic, but the Q-value for chosen action is used on behalf of the Critic. To train the RNN based on RL, training signals for the RNN are derived by RL and the RNN is trained by supervised learning using the training signals. The training signal of Q-output for selected action a_t (other Q-output(s) are not trained) is

$$Q_{train,a_t,t} = r_{t+1} + \gamma \max_a Q_a(\mathbf{s}_{t+1}) \tag{1}$$

where r_{t+1} indicates the reward signal at the time $t+1$, and $Q_a(\mathbf{s}_{t+1})$ indicates Q-output for action a when the sensor signal vector $\mathbf{s}$ at time $t+1$ is the input of the network. TD-error is defined as

$$\hat{r}_t = Q_{train,a_t,t} - Q_{a_t}(\mathbf{s}_t) = r_{t+1} + \gamma \max_a Q_a(\mathbf{s}_{t+1}) - Q_{a_t}(\mathbf{s}_t). \tag{2}$$

The training signals for the Actor outputs are

$$\mathbf{m}_{train,t} = \mathbf{m}(\mathbf{s}_t) + \hat{r}_t \mathbf{rnd}_t \tag{3}$$

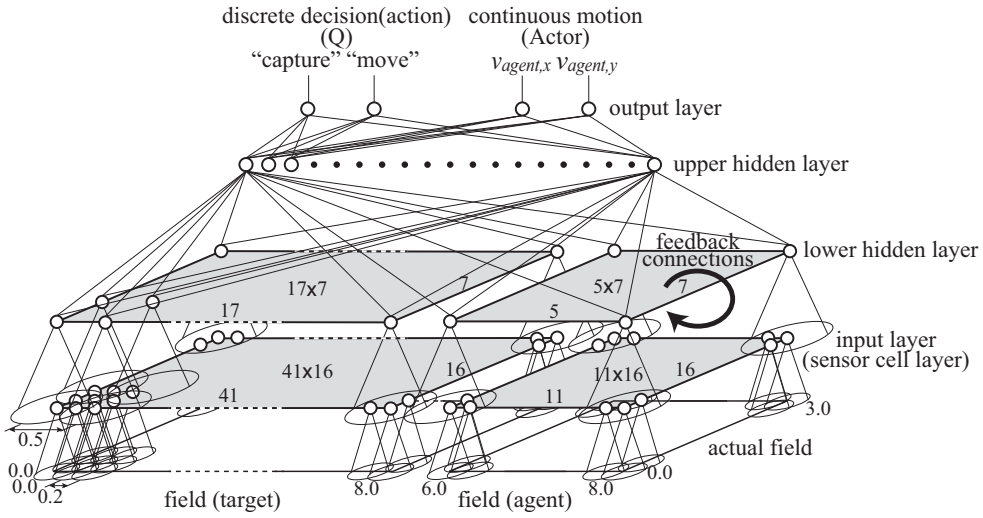
where $\mathbf{m}(\mathbf{s}_t)$ is the Actor output vector when $\mathbf{s}_t$ is the input of the RNN, and $\mathbf{rnd}_t$ is the exploration vector as mentioned. Then $Q_{train,a_t,t}$ and $\mathbf{m}_{train,t}$ are used as training signals, and the RNN with the input $\mathbf{s}_t$ is trained once according to BPTT (Error Back Propagation Through Time) [19]. Here, the sigmoid function whose value ranges from -0.5 to 0.5 is used as the output function of each hidden or output neuron. Therefore, to adjust the value range, the range of the output neuron $[-0.5, 0.5]$ is shifted to that of the actual Q-value $[-0.2, 0.8]$. The Actor outputs are used directly without any transformation. What should be emphasized is that the learning is very simple and general, and as the readers will notice, no special learning for the given task is applied.

III. Simulation

A. Invisible Target Capture Task

The task is described in Fig. 1. There is a field with the size of 8.0×3.0. In each episode, a target starts from the left end of the field. Its initial y-coordinate, $p_{y,0}$, the x-component of its velocity, $v_{x,0}$ and its moving direction from the x-axis, θ_0, are randomly chosen as in the figure. Here, the velocity means the move in one step. The target moves straight, and when it reaches $y = 0.0$ or $y = 3.0$, it bounces at a wall, and v_x and v_y are changed to 0.9 times and -0.8 times respectively. An agent is initially located at $(7.0, 1.5)$. When it chooses "move" action, it moves according to the sum of the Actor output vector $\mathbf{m}(\mathbf{s}_t)$ and exploration vector $\mathbf{rnd}_t$. One move is limited in ± 0.4 in each of x- and y-directions that is the half of the maximum v_x of the target. When it chooses "capture" action, it does not move and captures the target at the next step. If $dist < 1.0$ and $p_x > 6.0$ where $dist$ is the distance between the agent and target and p_x is the target's x-coordinate, the agent can get a reward, and the episode is terminated. The reward value is given as $0.7 * (1.0 - dist^2)$ considering the value range, but when the target is captured at $6.0 < p_x \leq 6.1$, it is discounted to $(p_x - 6.0) * 10$ times for smooth change in the reward value along p_x. If the agent chooses "capture" action when $p_x < 6.0$ or $dist > 1.0$, or does not choose it until $p_x > 8.0$, a small penalty -0.1 is imposed. In these cases, the episode is not terminated until $p_x > 8.0$ for acceleration of learning, but it is terminated in the test phase after learning.

For prediction to be required to accomplish the task, sometimes the target becomes invisible on the visual field.



layer	No. of neurons	initial connection weights
output	4	random -0.1 to 0.1
upper hidden	30	0.0
lower hidden	154	(from input layer) locally connected (self-feedback) 4.0 (other-feedback) 0.0
input	832	—

Fig. 2. Actor-Q-learning with a recurrent neural network (RNN). Please refer to the body text in detail. Number of neurons and initial weights in each layer are shown in the right table.

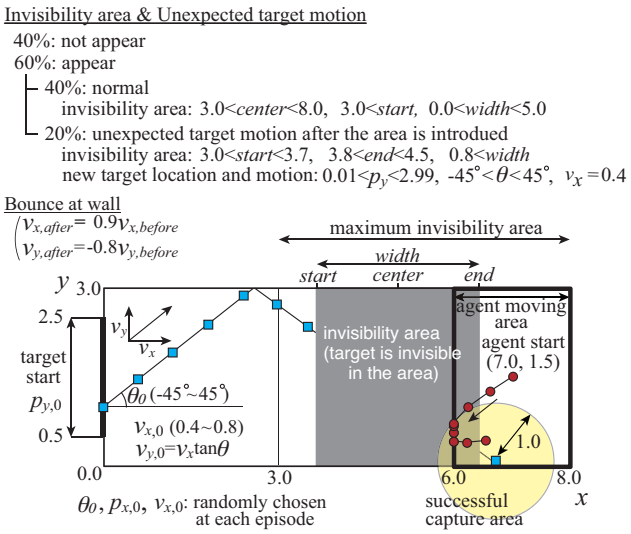


Fig. 1. "Invisible target capture" task.

As written in the upper part of Fig. 1, with the probability of 60%, an invisibility area, whose parameters such as $width$ are randomly decided, but the $start$ of the area is not less than 3.0, appears. When the invisibility area finishes, the target appears again in the visual field, but when the end of the area is large, the agent has to capture before it appears. In such cases, the agent has to predict the target location from the trajectory before it disappears. However, since the agent does not know where the invisibility area covers in advance, it cannot know when the target disappears or appears.

To examine the emergence of switching between two strategies, contradiction to the prediction before disappearance occurs with the probability of 20%. In this case, the invisibility area appears in the range of $3.0 \leq x \leq 4.5$ on the condition of $width > 0.8$. When the target appears again from the invisibility area, its y-coordinate p_y and moving direction θ are chosen randomly, and v_x is set to the minimum of 0.4. Accordingly, the prediction from the target movement before the disappearance is not helpful, and the agent has to observe the new target movement and predict the future state again.

Figure 2 shows the input, output and architecture of the RNN. Two visual sensors fixed in the world coordinate system are assumed; one is for the target and the other is for the agent. The sensor cells are arranged on the two-dimensional plane with the interval of 0.2 for each of x- and y-direction, and each cell has a small receptive field. The output of each cell is calculated as a Gaussian function whose input is the target or agent location, and the signals from the cells are the input of the RNN. So, there are $41 \times 16 = 656$ inputs for the target location and $11 \times 16 = 176$ inputs for the agent location, and 832 input signals in total.

The lower hidden neurons are also arranged on two-dimensional plane, and each neuron initially has a local responsible area, which means that the initial weights are calculated by a Gaussian function of the distance between the centers of the hidden and input neurons in the plane as shown in Fig. 2. Each of these neurons also has feedback connections from all the neurons in this layer as an Elman network. The initial self-feedback and other feedback connection weights are 4.0 and 0.0 respectively for effective and non-divergent propagation of error signal in BPTT because 4.0 is the inverse of the maximum derivative of the sigmoid function. The initial weights between the lower and upper hidden neurons are all 0.0 that makes all the outputs of NN are 0.0 initially not depending on the inputs because the output of each neuron is 0.0 when the sum of its inputs is 0.0. The initial weights between the upper hidden and output neurons are random values in ± 0.1, and the upper hidden neurons are expected to integrate the information about the target and agent through learning.

In Actor-Q-learning, the outputs are divided into Q-value outputs and Actor outputs as mentioned. Here, there are two outputs for Q-values, each of which is corresponding to "capture" or "move" action (decision). One action is selected according to Boltzmann selection [20] referring to the Q-values after the linear transformation as mentioned. There are also

two outputs for Actors, $v_{agent,x}$ and $v_{agent,y}$, each of which represents continuous-valued agent move in x- or y-direction. As an exploration factor, a uniform random number is added to each Actor output, and the agent moves according to the sum. Parameters used in this learning are shown in Table I.

TABLE I. PARAMETERS USED IN LEARNING. '->' MEANS THE VALUE IS REDUCED LINEARLY DURING LEARNING.

learning rate (for feedback) (for others)	0.02->0.01 0.1->0.05	temperature for Boltzmann selection	0.05->0.025
discount factor	0.96	exploration (uniform random number) size for actor	±0.5->±0.15

B. Results

Figure 3 shows the learning curves. Fig. 3(a) and Fig. 3(b) show the change in the success rate and that in the average obtained reward respectively. The exploration factors change as the number of episodes in both discrete actions and continuous motions during learning, and they influence the performance. Then, to see the influence of the factors, the performance when using the weights after learning is also plotted in each graph. In Fig. 3(b), two kinds of learning curve are shown. One shows the average of obtained reward over only successful episodes and the other shows that over all the episodes in each 1,000 episodes. The success rate is almost 0 at first and in most cases, the agent chose "capture" action before the target reaches the range of $p_x > 6.0$. However, the rate increases gradually, and finally it reaches almost 100%. The average over successful episodes is around 0.4 at first. When one location is randomly chosen in the success area around the agent that is a circle with the radius of 1.0, the expected reward is 0.35 that is the half of the maximum reward. As learning progresses, the reward is getting larger gradually. After learning, the performance was observed in 10,000 episodes with a random initial states. The number of failure episodes was 4. In one case, "capture" was chosen before the target comes into $x > 6.0$, and in the other three cases, the distance from the target to the agent was more than 1.0. The average reward was 0.656, and the average distance between the target and agent was 0.21.

The learning is performed 10 times by changing the random number sequence that is used to decide the initial connection weights, target motion and invisibility area at each episode, and also the exploration of the agent at each step. After learning, the average number of failures in 10,000 episodes is 9.0 (0.09%, SD 5.2) and the average reward is 0.652 (SD 0.003). When the local initial connection as shown in Fig. 2 is not employed and just using random numbers to decide them, the average number of failures is 57.6 (SD 20.7) and the average reward is 0.630 (SD 0.007). On the other hand, when the initial self-feedback connection weights are 0.0 instead of 4.0, learning is very slow and the number of failures is 1,687. When the regular no-traceback BP (Back Propagation) is used for learning of RNN instead of BPTT, the number of failures is 2,312. In these two cases, just one simulation is done with the same initial connection weights to the case whose results are shown in this paper. These results suggest that the local initial weights between lower hidden layer and input layer and also error-signal propagation to the past state by BPTT through 4.0 self-feedback connection weights are good for learning.

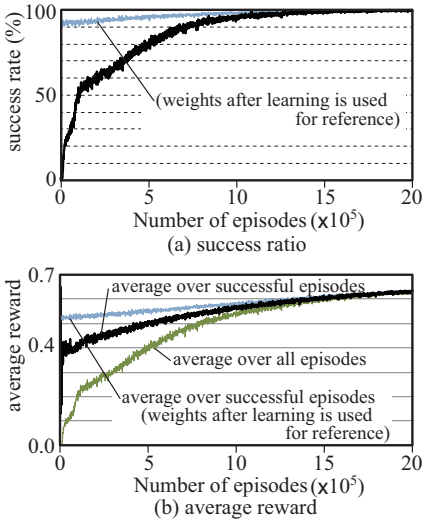


Fig. 3. Learning curves. (a) Success rate, (b) Average rewards

Figure 4(a) shows sample trajectories in 4 cases after learning when the size of the invisibility area is the maximum. In the case 1, the x-component of the target velocity $v_{x,0}$ is the minimum of 0.4, and the location where the agent has to get is close to the lower wall at $y = 0$. In the case 2, the initial target location $p_{y,0}$ and $v_{x,0}$ are the same, but the moving direction θ is different, and so the agent has to go toward the upper wall at $y = 3$. In the case 3, $p_{y,0}$ and θ_0 are the same as the case 1, but $v_{x,0}$ is the maximum of 0.8, so the timing to capture the target is completely different. In the case 4, the target goes parallel with the wall with the maximum velocity. The target location before disappearance is close to the case 3, but the capture position is completely different. In all the cases, it is seen that the agent approaches the target, and successfully captures it. It is interesting that the agent goes to the left end of the agent movable area at $x = 6.0$ at first, and when the target comes close, the agent goes to right ahead of the target. Same tendency can be seen in other simulation runs. That might be because the agent can capture the target in smaller number of steps by capturing it at smaller x-coordinate around $x = 6.0$ than around $x = 8.0$. One more reason might be that even if the agent does not choose "capture" action by the exploration factor in Boltzmann selection, it can capture the target at the next step by moving the same direction as the target.

Figure 4(b) shows the change of Q values that are two of the RNN outputs in one episode for the cases of 1 and 3 where only the target velocity $v_{x,0}$ is different. In either case, the Q-value for "capture" is far smaller than that for "move" action at first by the penalty when the agent chooses the "capture" action at too early timing. The Q-value for "move" increases gradually, but the Q-value for "capture" rises up suddenly, and becomes larger than that for "move" action finally. By the penalty at $p_x > 8.0$, the Q-value for "move" decreases at last. The larger Q-value at each step increases gradually, and the slope is close to the ideal one by the discount rate 0.96. It is seen that even though the initial Q-value is around 0.4, Q-value for "move" becomes different soon depending on $v_{x,0}$. That means that the agent can know the difference of capture timing soon after the target starts.

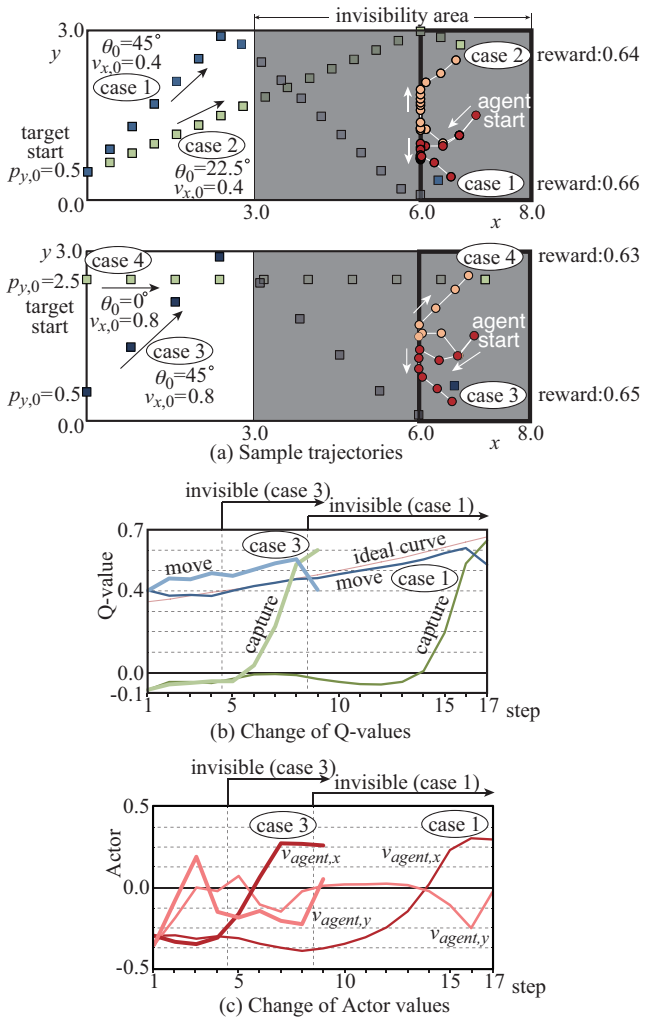


Fig. 4. (a) Sample trajectories of agent and target after learning, and the change in (b)the Q-values and (c)Actor values for the two cases in (a) in which only the target velocity is different.

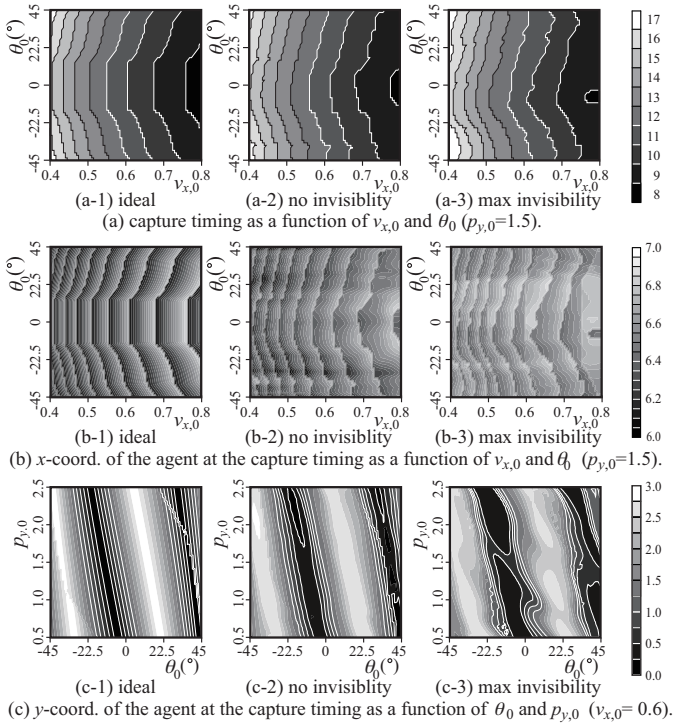


Fig. 5. (a) Capture timing and (b) agent's x-coordinate at the timing as a function of $v_{x,0}$ and θ_0 after learning for three cases including the ideal case. (c) Agent's y-coordinate at the capture timing as a function of θ_0 and $p_{y,0}$ also for the three cases. $v_{x,0}$, θ_0, and $p_{y,0}$ are x-component of velocity, moving direction, and y-coordinate of the agent in the initial state respectively. Each figure is made from the results in 81×61 test episodes after learning.

Figure 4(c) shows the Actor outputs. As seen also in the trajectories in (a), x-component of the agent velocity $v_{agent,x}$ is negative at first, and before the Q-value for "capture" increases, it begins to increase. At the 17th step in the case 1 and at the 9th step in the case 3, since the "capture" action is chosen, the agent does not move actually, but the outputs are shown for reference. If "move" action is chosen by exploration, the Actor outputs are used. It is interesting that after the y component of the target velocity, v_y changes positive after the second bounce in Fig. 4(a), $v_{agent,y}$ increases even though the target was invisible.

Figure 5 shows (a) the capture timing and (b) x-coordinate at the capture timing for various x-component of the initial target velocity $v_{x,0}$ and moving direction of the target θ_0 for three cases. Ideal values are drawn in the first graph. In the second and third one, the actual values in the cases with no invisibility area and with the maximum invisibility area are drawn. In Fig. 5(c), y-coordinate at the capture timing for various θ_0 and initial y location of the target $p_{y,0}$ is shown. In Fig. (a), even though the length of the longest episode 17 is more than double of the shortest episode 8, (a-2) and (a-3) are very similar to (a-1). Especially, it is seen that due to the bounce of the target to the walls, the capture timing becomes late when the absolute value of the moving direction θ_0 is larger also in (a-2) and (a-3). The number of steps is a little bit larger in total than ideal especially in Fig. 5(a-3). That might be because the approach is not so precise due to the invisibility, and so the agent avoids capturing the target with penalty at ($p_x < 6.0$) or with discounted reward at ($6.0 \leq p_x \leq 6.1$).

From Fig. 5(b), it is known that even when the capture timing is the same, x-coordinate at the capture is depending on the target x velocity $v_{x,0}$. The reason of larger x-coordinate in total in (b-3) might be the same reason as the late capture timing in (a-3). In (b-3), vertical lines can be seen slightly at $v_x = 0.5, 0.6, 0.75$. That is the discontinuous change due to the difference in whether the target around $x = 3.0$ is in the invisibility area or not by the small difference of v_x. In Fig. 5(c), especially (c-3) looks different from (c-1), but the tendency of the y-coordinate change by the change in $p_{y,0}$ and θ_0 is reflected also in (c-3). It is also seen that the agent did not go around a wall ($y = 0$ or $y = 3$) to capture the target. That might be because the target sometimes moves contradicting to the prediction result, and also there are no possibility that the target goes to $y < 0.0$ or $y > 3.0$.

Finally, Fig. 6 shows a sample in which the target movement was changed when the target appeared again. In the case 5, the target movement was compatible with that before being invisible, and the agent could catch the target at the upper area. In the case 6, the target y velocity is inverted

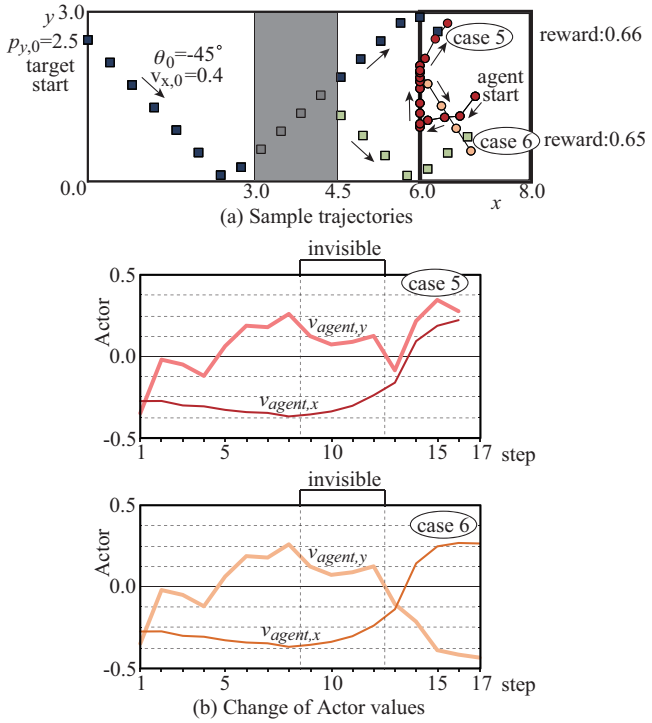


Fig. 6. Change in the agent's trajectory and Actor outputs caused by the change in the target motion.

while being invisible so that the agent has to go to lower area in contradiction to the first prediction. In these cases, the invisibility area begins at $x = 3$, and ends at $x = 4.5$. At first, the agent approached to $x = 6.0$ and after that went upward slightly, but does not approach so close to the upper wall. After the target appears again, the trajectories for the two cases diverge quickly, and the agent can reach the target in either case. Also in Fig. 6(b), it is seen that the Actor value for the agent y motion $v_{agent,y}$ after the agent appears again becomes positive in the case 5 and negative in the case 6. It is interesting that in the case 6, the capture timing is later than in the case 5. That might be because the agent could not reach close to the target at the 16th step, and chose to capture it at the 17th step after moving with positive $v_{agent,x}$. It is also suggested that the reason why the agent did not move so close to a wall at early steps might be that the incompatible target motion is also expected.

IV. Conclusion

In this paper, it is shown that in an "invisible-target capture" task, both prediction-required discrete decision and continuous motion could be learned only through reinforcement learning using a recurrent neural network without giving "prediction is necessary" or "what information should be predicted". The authors think of nothing except for this approach that can make an agent or robot acquire such flexible behaviors through learning only from rewards and punishments without giving any prior knowledge of the task. In this task, since the target object sometimes changes its moving direction randomly while being invisible, switching mechanism between strategies is required in general. However, that was acquired autonomously through learning without any additional architecture or technique. Such emergent property is what general parallel processing systems such as Subsumption architecture do not have, and the authors believe it is a key to solve the "Frame Problem" fundamentally.

Acknowledgment

This research was supported by JSPS Grant in-Aid for Scientific Research #19300070 & #23500245.

References

[1] Dennett, D. Cognitive Wheels : The Frame Problem of AI. *The Philosophy of Artificial Intelligence*, Margaret A. Boden, pp. 147-170, Oxford University Press, 1984.

[2] R. A. Brooks, Intelligence without Representation. *Artificial Intelligence*, **47**, pp.139-159, 1991.

[3] K. Shibata, Emergence of Intelligence through Reinforcement Learning with a Neural Network. *Advances in Reinforcement Learning*, InTech, pp.99-120, 2011.

[4] L.-J. Lin, & T. M. Mitchell, Reinforcement learning with hidden states, *From Animals to Animats 2*, pp. 271-280, MIT Press, 1993.

[5] A. Onat, H. Kita, & Y. Nishikawa, Q-Learning with Recurrent Neural Networks as a Controller for the Inverted Pendulum Problem, *Proc. of ICONIP 98*, pp. 837-840, 1998.

[6] H. Arie, T. Ogata, J. Tani, & S. Sugano, Reinforcement learning of a continuous motor sequence with hidden states, *Advanced Robotics*, **21** (10), pp. 1215-1229, 2007.

[7] A. Onat, H. Kita & Y. Nishikawa, Reinforcement learning of dynamic behavior by using recurrent neural networks, *Artificial Life Robotics*, **1**, pp. 117-121, 1997.

[8] B. Bakker, V. Zhumatiy, G. Gruener & J. Schmidhuber, A Robot that Reinforcement-Learns to Identify and Memorize Important Previous Observations, *Proc. of IROS 2003*, pp. 430-435, 2003.

[9] K. Shibata, Discretization of Series of Communication Signals in Noisy Environment by Reinforcement Learning, *Adaptive and Natural Computing Algorithms (Proc. of ICANNGA'05)*, pp. 486-489, 2005.

[10] H. Utsunomiya & K. Shibata, Contextual Behavior and Internal Representations Acquired by Reinforcement Learning with a Recurrent Neural Network in a Continuous State and Action Space Task, *Advances in Neuro-Information Processing*, LNCS, **5506**, pp. 755-762, 2009

[11] K. Goto & K. Shibata, Acquisition of Deterministic Exploration and Purposive Memory through Reinforcement Learning with a Recurrent Neural Network, *Proc. of SICE Annual Conf. 2010*, 2010.

[12] K. Shibata & H. Utsunomiya, Discovery of Pattern Meaning from Delayed Rewards by Reinforcement Learning with a Recurrent Neural Network, *Proc. of IJCNN. 2011*, pp. 1445-1452,2011.

[13] J. Schmidhuber, Reinforcement learning in Markovian and non-Markovian environments, In D. S. Lippman, J. E. Moody, and D. S. Touretzky (eds), *Advances in Neural Information Processing Systems 3*, pp. 500-506, Morgan Kaufmann, 1991.

[14] J. Tani, Learning to generate articulated behavior through the bottom-up and the top-down interaction process, *Neural Networks*, **16**(1), pp. 11-23, 2003.

[15] K. Goto, & K. Shibata, Emergence of prediction by reinforcement learning using a recurrent neural network, *Journal of Robotics*, **2010**, Article ID 437654, 2010.

[16] C. J. C. H. Watkins, Learning from Delayed Rewards, *PhD thesis*, Cambridge University, Cambridge, England, 1989.

[17] K. Shibata, T. Nishino & Y. Okabe Active Perception and Recognition Learning System Based on Actor-Q Architecture, *Systems and Computers in Japan*, **33**(14), pp. 12-22, 2002.

[18] A. G. Barto, et al., Neuronlike adaptive elements can solve difficult learning control problems, *IEEE Trans. on Systems, Man, and Cybernetics*, **13**(5), pp. 834-846, 1983.

[19] D. E. Rumelhart, et al., Learning Internal Representation by Error Propagation, *Parallel Distributed Processing*, MIT Press, **1**, pp. 318-364, 1986.

[20] R. S. Sutton & A. G. Barto, *Reinforcement Learning: An Introduction*, A Bradford Book, The MIT Press,1998