*ICIC Express Letters*
*Part B: Applications*
Volume **8**, Number **4**, April **2017**

 ISSN 2185-2766
pp. **727–733**

# A STUDY ON BEHAVIOR ACQUISITION OF MOBILE ROBOT BY DEEP Q-NETWORK

Hikaru Sasaki[1], Tadashi Horiuchi[2] and Satoru Kato[3]

[1]Advanced Engineering Faculty
[2]Department of Control Engineering
[3]Department of Information Engineering
National Institute of Technology, Matsue College
14-4 Nishi-ikuma, Matsue, Shimane 690-8518, Japan
{ s1509; horiuchi; kato }@matsue-ct.ac.jp

Received September 2016; accepted December 2016

Abstract. *Deep reinforcement learning is a framework which combines deep learning and reinforcement learning. Deep Q-network (DQN), recently proposed by V. Mnih et al. is a successful method which uses convolutional neural network (CNN) in order to approximate the action-value function in Q-learning, a widely-used reinforcement learning algorithm. DQN can learn successful policies directly from high-dimensional sensory inputs using reinforcement learning in Atari 2600 video games. In this research, we apply DQN to robot behavior learning in the simulation environment. We realize that the mobile robot itself learns to acquire good behaviors such as avoiding the walls and moving along the center line by using high-dimensional visual information as input data. The mobile robot has three cameras: one is in front of the robot and others are in left and right directions of the robot. Through the simulation experiment using the robot simulator Webots, we confirm the effectiveness of this method.*
**Keywords:** Deep learning, Reinforcement learning, Deep Q-network, Robot behavior learning, Convolutional neural network

1. **Introduction.** Deep learning is a framework of machine learning using deep neural network by which automatic feature extraction is enabled from raw data such as image data and acoustic data. Convolutional neural network (CNN) [1, 2] is attracting considerable attention as one of the deep learning methods. CNN is a family of multi-layer neural networks in which the hidden units have local receptive fields. CNN consists of feature extraction part and classification part, both of which are trainable by back-propagation method.

Deep Q-network (DQN), recently proposed by V. Mnih et al. [3, 4], is a successful framework which combines reinforcement learning with CNN. DQN can learn successful policies directly from high-dimensional sensory inputs using reinforcement learning in the classic Atari 2600 video games. In DQN, CNN is used in order to approximate the action-value function called Q-function in Q-learning [5] which is a widely-used reinforcement learning algorithm.

In this research, we construct the simulation environment for two-wheeled mobile robot and then we apply DQN to the robot learning in the simulation environment. We realize the mobile robot learns to acquire good behaviors such as avoiding the wall and moving along the center line by using high-dimensional visual information as input data. In this research, we propose a modified method of DQN which stores the best target network parameters so far and replace the target network to the best target network so far if the performance of DQN using the updated target network decreases suddenly in performance evaluation (test run phase). Through the simulation experiment using robot simulator Webots, we confirm the effectiveness of our proposed method for robot behavior learning.

2. **Convolutional Neural Network (CNN).** CNN is a variant of multi-layer neural networks particularly designed for use on two-dimensional data such as images and videos [2]. In CNN, the hidden units have local receptive fields as in the primary visual cortex and the weights are tied or shared across the image in order to reduce the number of parameters. CNN consists of feature extraction part and classification part, both of which are trainable by back-propagation method. Figure 1 illustrates network structure of CNN. In the feature extraction part, there are several pairs of convolution process and pooling process.

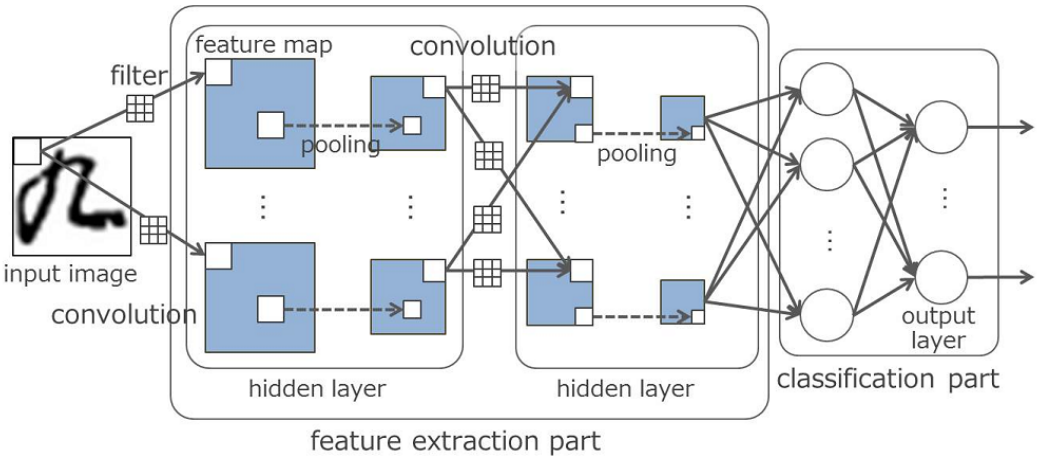



FIGURE 1. Network structure of CNN

**[Convolution process].** First of all, the input image is convolved with a set of $N$ small filters whose coefficients (weights) are trained by supervised learning. By convolution process using $N$ small filters, $N$ "feature maps" are obtained. A feature map is obtained by convolving the input image with a linear filter, adding a bias term and then applying a non-linear function.

**[Pooling process].** This first stage is followed by pooling process that reduces the dimensionality and offers robustness to spatial shifts. In the pooling process, several typical methods have been proposed. For example, max-pooling is an operation to take a maximum value in each $2 \times 2$ region. Average-pooling is an averaging operation in each $2 \times 2$ region. In this research, we use max-pooling method. The results of pooling process become inputs to the next convolution process and then convolved with a set of small filters again.

**[Classification part].** The outputs of the last pooling process of feature extraction part are forwarded to the fully-connected multi-layer perceptron that produces the final classification output of the system.

**[Learning algorithm].** In CNN, all weights of the fully-connected multi-layer perceptron in classification part and coefficients of the filters in feature extraction part are trained by the standard back-propagation algorithm.

3. **Deep Q-Network (DQN).** DQN, recently proposed by V. Mnih et al. [3, 4], is a successful framework which combines reinforcement learning with CNN. DQN can learn successful policies directly from high-dimensional sensory inputs using end-to-end reinforcement learning in the classic Atari 2600 video games.

In reinforcement learning, we consider the learning tasks in which the agent interacts with an environment through the sequence of state observations, actions and rewards. The goal of the agent is to select actions to maximize the cumulative future reward [6].

CNN is used in DQN in order to approximate the following optimal action-value function called optimal Q-function:

$$Q^*(s,a) = \max_a \boldsymbol{E}\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots | s_t = s, a_t = s, \pi\right] \tag{1}$$

which means the maximum sum of reward $r_t$ discounted by $\gamma$ at each time step $t$, based on a policy $\pi = P(a|s)$ after making a state observation $s$ and taking an action $a$. CNN's output that is used in DQN is Q-value for an action taken by agent. Figure 2 shows network structure of CNN in DQN.

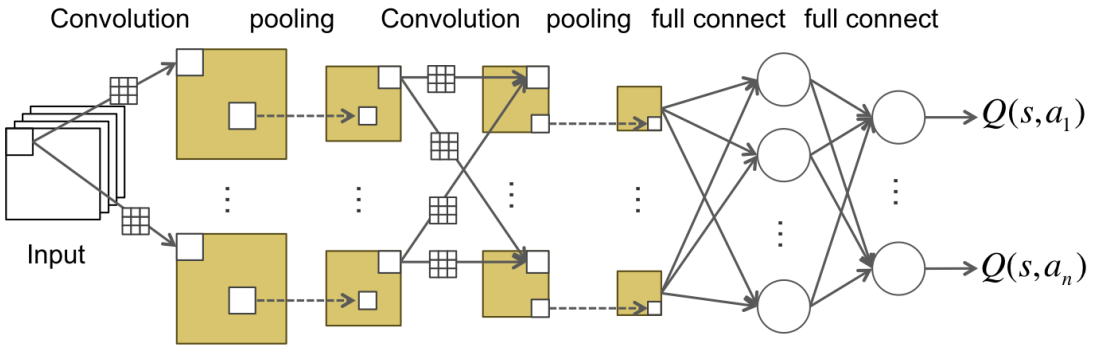



FIGURE 2. Network structure of CNN in DQN

Reinforcement learning is known to be sometimes unstable or even to diverge when a nonlinear function approximator such as a neural network is used to represent the action-value function (Q-function). There are several reasons for this instability such as 1) the correlations present in the sequence of state observations; 2) small updates to Q-value may significantly change the policy and therefore change the data distribution.

In order to overcome the above problems, DQN uses the method termed experience replay. To perform experience replay, the agent's experiences $e_t = (\phi_t, a_t, r_t, \phi_{t+1})$ at each time step $t$ are stored in the data set $D = \{e_1, \ldots, e_t\}$. During learning, we apply Q-learning updates on samples of experience $(\phi_j, a_j, r_j, \phi_{j+1})$ drawn uniformly at random from the data set. The Q-learning update at iteration $i$ uses the following loss function:

$$L_i(\theta_i) = \boldsymbol{E}\left[\left(r + \gamma \max_{a'} Q\left(s', a'; \theta_i^-\right) - Q(s, a; \theta_i)\right)^2\right] \tag{2}$$

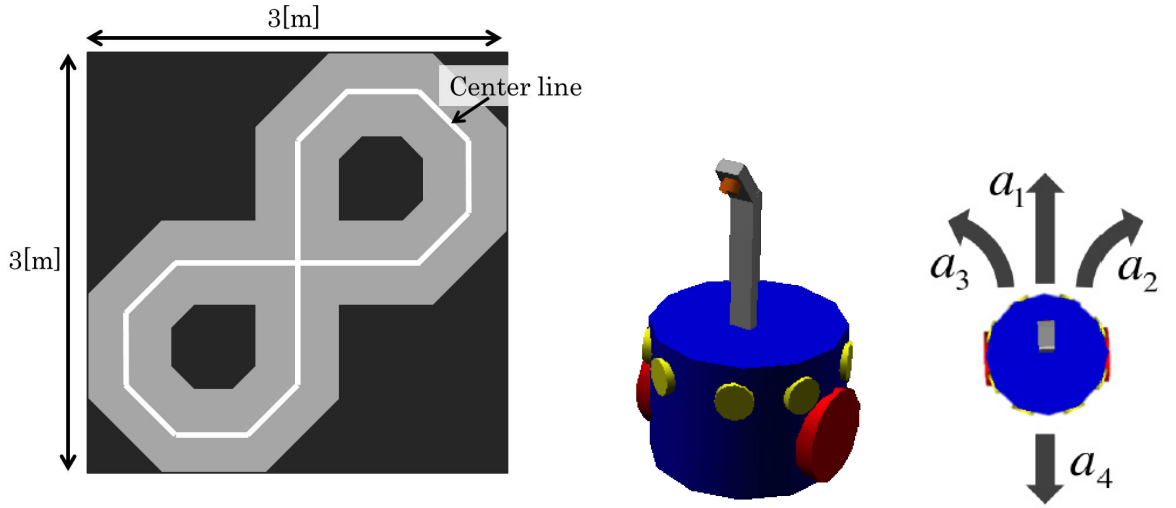
in which $\gamma$ is the discount factor and $\theta_i$ are the network parameters of the Q-network at iteration $i$, and $\theta_i^-$ are the network parameters used to compute the target at iteration $i$. The full algorithm of deep Q-learning with experience replay is shown in Figure 3. In Figure 3, $\epsilon$ denotes a small portion of probability for the random action selection in $\epsilon$-greedy method, $D$ is a data set to store the set of transition $(\phi_t, a_t, r_t, \phi_{t+1})$ for the experience replay, $j$ means the ordinal number in the set of minibatch data, and $C$ denotes the interval to update the network parameters (the network parameters are updated every $C$ steps).

4. **Application of DQN to Robot Learning.** In this research, we apply DQN to robot behavior learning in the simulation environment. We realize the behavior acquisition of two-wheeled mobile robot in the simulation environment. For this purpose, we use Cyberbotics's robot simulator Webots.

**Algorithm 1.** Deep Q-learning with experience replay

**for** episode $= 1, M$ **do**
  Initialize sequence $s_1 = x_1$ and preprocess $\phi_1 = \phi(s_1)$
  **for** $t = 1, T$ **do**
    With probability $\epsilon$ select a random action $a_t$
    otherwise select $a_t = \text{argmax}\, Q(\phi(s_t), a; \theta)$
    Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in the data set $D$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
    Set $y_j = \begin{cases} r_j \text{ if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) \text{ otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters $\theta$
    Every $C$ steps reset $\hat{Q} = Q$
  **end for**
**end for**
**end**

FIGURE 3. Algorithm of deep Q-learning with experience replay




(a) Simulation environment

(b) Two-wheeled mobile robot

(c) Candidate actions of mobile robot

FIGURE 4. Simulation environment and two-wheeled mobile robot

**[Problem setting].** The target task for the agent is to acquire the behavior of two-wheeled mobile robot to move without collision to the walls in the problem environment as shown in Figure 4(a). Figure 4(b) illustrates the overview of two-wheeled mobile robot. The size of the problem environment is 3[m] × 3[m] and the width of the road is 60[cm]. The center line is painted by gray color in the center of the road. The shape of road is like a slanted digit '8' as shown in Figure 4(a).

The specifications of two-wheeled mobile robot are shown in Table 1. The robot has a camera on top of the robot in order to get a front view. This camera outputs the image with $50 \times 30$ [pixels] size. Figure 5 illustrates four examples of camera images of the robot. In this figure, the walls are colored with black, the floor is colored with gray, the center line is colored with white, and the body of robot is colored with dark gray.

TABLE 1. Specification of two-wheeled mobile robot

| | |
|---|---|
| Diameter of body | 16[cm] |
| Height | 12[cm] |
| Diameter of wheels | 5[cm] |
| Camera | 1 direction |
| Distance sensors | 8 directions |

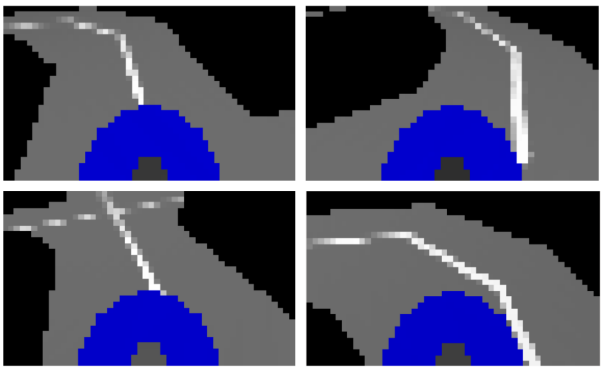


FIGURE 5. Examples of camera image from top of the robot

The robot also has distance sensors with 8 directions, which enable to measure the distance between robot and walls. The range of the sensor values is between 0 and 1,000. When the robot is far away from the walls, the sensor value is 0. On the other hand, when the robot is very near to the wall, the sensor value is almost 1,000. The measurable range of this sensor is between 70[cm] and 1.0[cm].

**[Definition of action, state and reward].** The robot takes an action at each time step among four candidates:

$$a_t = \begin{cases} a_1 & \text{(go straight)} \\ a_2 & \text{(turn right)} \\ a_3 & \text{(turn left)} \\ a_4 & \text{(go backward).} \end{cases}$$

The action "go straight" is realized by rotating both wheels clockwise and the robot moves forward about 5.0[cm] by this action. The action "rotate clockwise" is realized by rotating the left wheel clockwise and rotating the right wheel counter-clockwise. By this action, the robot moves about 7.0[cm] and turns around about 13.2[deg] to the clockwise direction. The action "rotate counter-clockwise" is realized by opposite motions of the two wheels. Taking an action at each time step means 1 step action in this research.

The state observation is defined using the camera images and 8 distance sensors of the robot. Suppose the camera image at time step $t$ is $x_t$, we get $\phi_t = \phi(x_t)$ by applying the preprocessing function $\phi$. Here, preprocessing function transforms the RGB color image to the gray-scaled image. The state $s_t$ is defined by using the most recent two frames $\phi_t$, $\phi_{t-1}$. Hence, $s_t = \{\phi_t, \phi_{t-1}\}$.

The reward $r_t$ at time step $t$ is defined by using the values of 8 distance sensors of the robot as follows:

$$r_t = \begin{cases} 10 & \text{(when robot is far away from the wall)} \\ -1 & \text{(when robot selects the backward action)} \\ -20 & \text{(when robot collides to the wall)} \\ 0 & \text{(otherwise).} \end{cases}$$

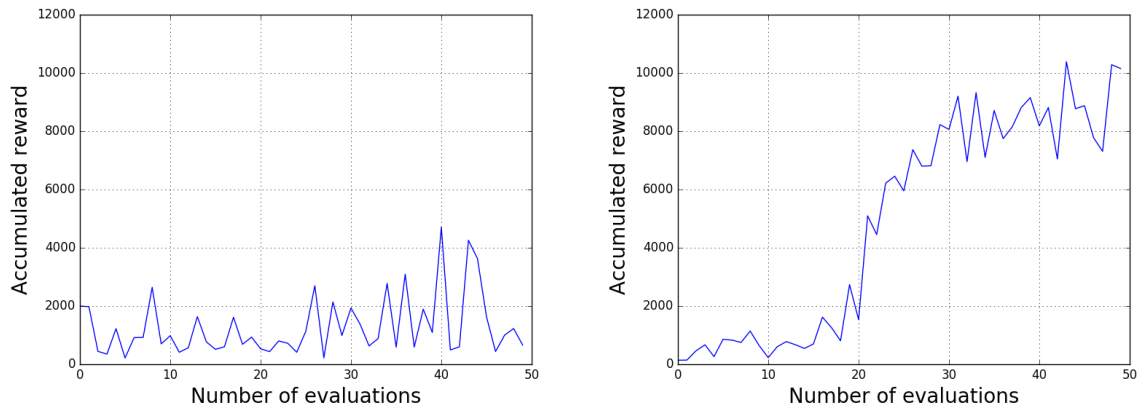
The structure of CNN is as follows: there are two hidden layers each of which has $8 \times 8$ filters and $2 \times 2$ max-pooling. The number of filters is 32 both in the first convolution layer and in the second convolution layer. The inputs are four gray-scaled images of $50 \times 30$ [pixels] size. The number of nodes in the hidden layer of classification part is 256. The number of nodes in the output layer is 4, which equals the number of action candidates of the robot.

**[Modification of DQN for robot learning].** In the original DQN as shown in Figure 3, every $C$ steps we clone the network $Q$ to obtain a target network $\hat{Q}$ and use $\hat{Q}$ for generating the targets $y_j$ for the following $C$ updates to $Q$. This makes the algorithm more stable compared with standard Q-learning. Generating the targets $y_j$ using an older set of network parameters adds a delay between the time an update to $Q$ is made and the time the update affects the targets $y_j$, making oscillations much more unlikely.

However, in our problems of the robot behavior learning, the original DQN shows unstable learning performance. In this research, we propose a modified method of DQN which stores the best target network parameters so far and replace the target network $\hat{Q}$ to the best target network so far $\hat{Q}_{best}$ if the performance of DQN using the updated target network $\hat{Q}$ decreases suddenly less than 30% in performance evaluation (test run phase).

5. **Experiment.** In the problem setting described in the previous section, we execute the experiment using robot simulator Webots. The action selection is made by $\epsilon$-greedy method. The value of $\epsilon$ is 0.2 in this experiment. During the first 10 thousand steps, the robot only collects the pairs of experienced states, actions and rewards without learning. The data are stored in data set $D$. The size of data set $D$ is $100,000$ and the value of discount rate $\gamma$ is 0.99 in this experiment. After 10 thousand steps, the network parameters are updated using 32 pairs of data (minibatch size = 32) from data set $D$. At every $5,000$ steps of learning, the evaluation (test run) is performed for $5,000$ steps. Both in learning phase and test run phase, the robot restarts action selection from the far away point (near the center line) from the wall, when the robot collides to the wall.

We executed 5 sets of simulation experiments with different random seeds. Figure 6 illustrates the change of average accumulated reward in each test run for 5 sets of experiments. In this figure, the horizontal axis indicates the number of evaluations (test runs) and vertical axis means the accumulated reward at each test run. Compared with Figure 6(a) and Figure 6(b), we can confirm that our proposed method shows much better performance than the original DQN for the robot nagivation problem.




(a) Result without modification (original DQN) (b) Result with modification (our method)

FIGURE 6. Change of accumulated reward at test run (comparison of original DQN and our method)

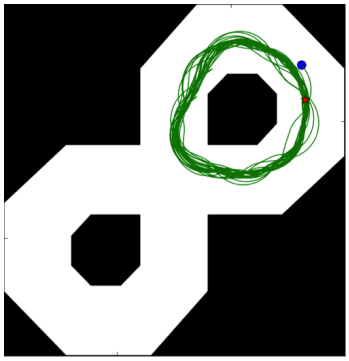


FIGURE 7. An example of acquired behaviors by our method

We also confirm that good behaviors such as avoiding the wall and moving along the center line are acquired by using the network with high reward. Figure 7 shows an example of acquired behaviors in the final stage of learning. Although it is not different from the behavior trajectory such as shape of '8' digit number which we expected, the accumulated reward in this run is very high.

6. **Conclusions.** In this research, we proposed a modified method of DQN for robot behavior learning. We constructed the simulation environment for two-wheeled mobile robot and then we apply the proposed method to the robot behavior learning in the simulation environment. By the proposed method, we realized the mobile robot learns to acquire good behaviors such as avoiding the wall and moving along the center line by using high-dimensional visual information as input data.

There are several future work such as: 1) acceleration of DQN learning method; 2) applying the proposed method to the real two-wheeled robot with Raspberry Pi and Web cameras.

**Acknowledgment.** This research was supported by JSPS KAKENHI Grant Number 15K00355 from the Japan Society for the Promotion of Science.

## REFERENCES

[1] Y. LeCun et al., Gradient-based learning applied to document recognition, *Proc. of the IEEE*, vol.86, no.11, pp.2278-2324, 1998.

[2] I. Arel et al., Deep machine learning – A new frontier in artificial intelligence research, *IEEE Computational Intelligence Magazine*, vol.5, no.4, pp.13-18, 2010.

[3] V. Mnih et al., Playing Atari with deep reinforcement learning, *Proc. of NIPS 2013 Deep Learning Workshop*, 2013.

[4] V. Mnih et al., Human-level control through deep reinforcement learning, *Nature*, vol.518, pp.529-533, 2015.

[5] C. J. Watkins and P. Dayan, Technical note: Q-learning, *Machine Learning*, vol.8, pp.279-292, 1992.

[6] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, The MIT Press, 1998.