Paper:

Q-Learning in Continuous State-Action Space with Noisy and Redundant Inputs by Using a Selective Desensitization Neural Network

Takaaki Kobayashi, Takeshi Shibuya, and Masahiko Morita

Faculty of Engineering, Information and Systems, University of Tsukuba
1-1-1 Tennodai, Tsukuba, Ibaraki 305-8573 Japan
E-mail: {takaaki@bcl.esys., shibuya@iit., mor@bcl.esys.}tsukuba.ac.jp
[Received May 21, 2015; accepted August 18, 2015]

When applying reinforcement learning (RL) algorithms such as Q-learning to real-world applications, we must consider the influence of sensor noise. The simplest way to reduce such noise influence is to additionally use other types of sensors, but this may require more state space – and probably increase redundancy. Conventional value-function approximators used to RL in continuous state-action space do not deal appropriately with such situations. The selective desensitization neural network (SDNN) has high generalization ability and robustness against noise and redundant input. We therefore propose an SDNN-based value-function approximator for Q-learning in continuous state-action space, and evaluate its performance in terms of robustness against redundant input and sensor noise. Results show that our proposal is strongly robust against noise and redundant input and enables the agent to take better actions by using additional inputs without degrading learning efficiency. These properties are eminently advantageous in real-world applications such as in robotic systems.

Keywords: reinforcement learning, function approximator, continuous state-action, sensor noise, redundant inputs

1. Introduction

Reinforcement learning (RL) [1] is a framework for trial-and-error-based interactive learning inspired by behavioral learning in animals. An agent working in this framework learns behavior for acquiring maximum rewards based on its experience in the environment. RL enables the agent to accomplish a task by simply rewarding the agent, thus acting as a promising control for systems such as robots in real-world environments.

In Q-learning [2], which is a widely used RL algorithm, the agent decides an action based on two functions – *action value* and *policy* – in which updating the action-value function improves the agent's behavior. With its proven convergence property [3] and its easy implementation, Q-learning is used frequently in applied research. When Q-learning is applied in an environment with continuous state-action space, the action-value function is generally approximated by using a function approximator [1, 4–9].

In real-world applications, Q-learning agents suffer from input uncertainty due to sensor noise. The simplest way to solve this problem would be to additionally use other types of sensors so that the noises from different sensors can be mutually cancelled. This, however, could enlarge state space and probably increase redundancy. Agents with conventional value-function approximators thus require more samples for their learning.

For this reason, a Q-learning function approximator should have the following properties when used in real-world applications:

- High robustness against input uncertainty
- High learning ability and robustness against redundant input

A recent study has demonstrated the superiority of selective desensitization neural networks (SDNNs) in function approximation [10]. Using an SDNN, Horie et al. [11] estimated hand motion speed from surface electromyograms, which are noisy and highly redundant input. Their results suggest that the SDNN is very suitable for function approximation involving noisy and redundant input. This makes the SDNN a strong candidate for approximating the action-value function in continuous state-action space with noisy and redundant state variables.

We propose an SDNN-based value-function approximator for Q-learning in continuous state-action space [12] and evaluate its performance in terms of robustness against redundant input [12] and sensor noise through a series of numerical experiments. These experiments use an acrobot swing-up task containing noisy and redundant input.

This paper is organized as follows: Section 2 explains the basic RL framework and the Q-learning algorithm, together with widely used ways for approximating the value-function. Section 3 proposes SDNN-C, a novel function approximation. Section 4 reviews the results of simulation experiments and Section 5 lists conclusions.

2. Background

2.1. Reinforcement Learning

RL is a trial-and-error-based interactive learning framework in which agents adjust their behavior to maximize their rewards in the environment.

At discrete time step t, an agent observes its own state, s_t, decides and takes action a_t, and obtains reward r_{t+1}. The agent learns suitable actions through this experience. Learning and deciding actions are implemented by using two functions—action value and policy.

Action-value function $Q(s,a)$ is defined as the expected total rewards from taking action a in state s. For a particular state-action pair, an action value is the value of the action-value function and indicates the desirability of the action. In RL, the agent updates its action-value function by using a learning algorithm such as Q-learning.

Policy provides probabilities for selecting possible actions in certain states. A simple, well-known policy is the ε-greedy policy, which encourages both exploration and exploitation. This policy frequently leads to selecting the action with the highest action value, but randomly with low probability ε.

2.2. Q-Learning

Q-learning [2, 3], a widely used RL algorithm, updates action-value function $Q(s_t,a_t)$ as

$$Q(s_t,a_t) \leftarrow (1-\alpha)Q(s_t,a_t) + \alpha\left(r_{t+1} + \gamma \max_{a' \in \mathscr{A}(s_{t+1})} Q(s_{t+1},a')\right), \quad (1)$$

where $\mathscr{A}(s)$ is a set of possible actions in state s. $\alpha \in [0,1]$ is a learning rate and $\gamma \in [0,1)$ is a discount rate.

Because $Q(s,a)$ is stored as a look-up table, conventional Q-learning cannot operate in continuous state-action space.

2.3. Function Approximators

Action-values in continuous state-action space are often implemented by using a function approximator, which represents the action-value by fewer parameters than a look-up table.

The multilayer perceptron, although a well-known general function approximator, is not suitable for directly approximating an action-value function because it is an offline algorithm. This means that an additional mechanism such as extra memory is required to store histories of states, actions, and rewards [4].

Function approximators based on a previous knowledge of action-value function landscapes have also been proposed [5, 6], but are applicable only if there is previous knowledge available.

Linear methods such as tile coding [1], the radial basis function network (RBFN) [1, 7] and fuzzy-based methods [8, 9] are often used to approximate a target function by using a weighted linear sum of features. Although all of these methods perform well without previous knowledge, they involve increasing difficulty as the numbers of experience required grow exponentially with the state space dimension [1]. This makes them unsuitable for the approach of adding redundant sensors.

In short, available function approximators are not easily applied to Q-learning in continuous state-action space with noisy and redundant input.

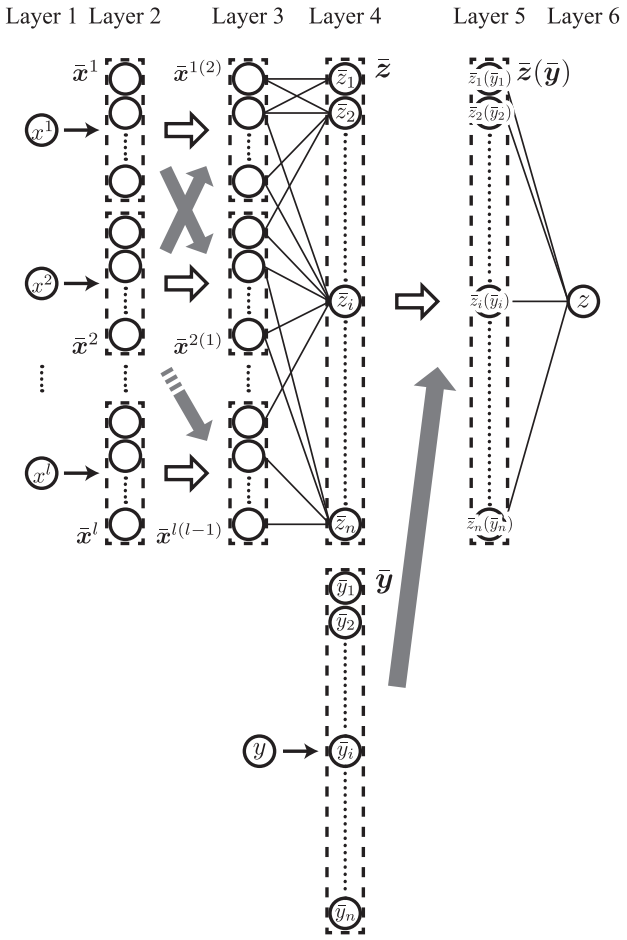


Fig. 1. SDNN-C structure.

3. Proposal

We propose introducing SDNN-C, an SDNN-based function approximator, for action-value functions.

3.1. SDNN-C Structure

SDNN-C has the six layers shown in **Fig. 1**.

Layer 1, which is the input layer, consists of l ($l \geq 2$) neural units, where l is the number of input signals. Each unit corresponds to one input signal. The input signal vector is denoted by $\boldsymbol{x}$.

Layer 2, which is a pattern coding layer for the state, consists of l neural groups, $(\bar{\boldsymbol{x}}^1, \ldots, \bar{\boldsymbol{x}}^l)$. This group's units' output are denoted $\bar{\boldsymbol{x}}^i$. Each group corresponds to one unit in layer 1 and outputs an m-dimensional binary

pattern on $\{-1,+1\}^m$, where m is the number of units in the group. The pattern is determined as follows:

1. Construct 9 random binary patterns $(\bar{\boldsymbol{p}}_1, \bar{\boldsymbol{p}}_2, \ldots, \bar{\boldsymbol{p}}_9)$.
2. Construct 44 binary patterns by interpolating between $\bar{\boldsymbol{p}}_i$ and $\bar{\boldsymbol{p}}_{i+1}$ $(i = 1,2,\ldots,8)$.
3. Divide the output range of the corresponding unit in layer 1 into 360 equal subranges.
4. Assign 360 patterns to each subrange, omitting $\bar{\boldsymbol{p}}_9$.

Layer 3, which is a state desensitization layer, consists of $l \times (l-1)$ neural groups, $\{\bar{\boldsymbol{x}}^{\mu(\nu)} \mid \mu,\nu = 1,\ldots,l;\ \mu \neq \nu\}$, where $\bar{\boldsymbol{x}}^{\mu(\nu)}$ denotes $\bar{\boldsymbol{x}}^{\mu}$ desensitized by $\bar{\boldsymbol{x}}^{\nu}$. Desensitization neutralizes outputs of units in $\bar{\boldsymbol{x}}^{\mu}$ when the corresponding unit in $\bar{\boldsymbol{x}}^{\nu}$ outputs -1, regardless of the unit in $\bar{\boldsymbol{x}}^{\mu}$. The output of the k-th unit is given by

$$\bar{x}_k^{\mu(\nu)} = \frac{1 + \left(\sigma_{\mu\nu}\bar{\boldsymbol{x}}^{\nu}\right)_k}{2}\,\bar{x}_k^{\mu},$$

$\bar{x}_k^{\mu}$ is the output of the k-th unit of $\bar{\boldsymbol{x}}^{\mu}$ and $\sigma_{\mu\nu}$ is an m-dimensional permutation matrix that randomly permutes units of $\bar{\boldsymbol{x}}^{\nu}$. For simplicity, a vector containing all $\bar{\boldsymbol{x}}^{\mu(\nu)}$ for μ and ν $(\mu \neq \nu)$, is denoted by $\bar{\boldsymbol{x}}^{SD}$.

Layer 4 consists of two different neural groups, i.e., $\bar{\boldsymbol{y}}$ and $\bar{\boldsymbol{z}}$. $\bar{\boldsymbol{y}}$ is a pattern coding group for action y, and has n units. It outputs an n-dimensional binary pattern representing the action determined by the following procedure:

1. Construct $(n-n'-1)$ patterns, in which n' represents the number of 1s. The i-th element of the k-th pattern is given by

$$\bar{q}_{k,i} = \begin{cases} +1 & (\text{if } k \leq i < k+n') \\ -1 & (\text{otherwise}) \end{cases}.$$

2. Divide the range of the action variable into $(n-n'-1)$ equal subranges.
3. Assign these patterns to each subrange.

$\bar{\boldsymbol{z}}$ is a hidden layer with n units. Every unit in $\bar{\boldsymbol{z}}$ is connected to every unit in layer 3. The output of the i-th unit of $\bar{\boldsymbol{z}}$ is given by

$$\begin{aligned}\bar{z}_i &= \phi\left(\boldsymbol{w}_i^{\top}\bar{\boldsymbol{x}}^{SD} - w_{i0}\right) \\ &= \phi\left(\sum_{j=1}^{ml(l-1)} w_{ij}\bar{x}_j^{SD} - w_{i0}\right),\end{aligned}$$

where

$$\phi(u) = \begin{cases} 1 & (u > 0) \\ 0 & (\text{otherwise}) \end{cases},$$

$\bar{x}_j^{SD}$ is the j-th element of $\bar{\boldsymbol{x}}^{SD}$, w_{ij} is a synaptic weight between the j-th unit in layer 3 and the i-th unit in $\bar{\boldsymbol{z}}$ of layer 4 and w_{i0} is a threshold. The quantity $\left(\boldsymbol{w}_i^{\top}\bar{\boldsymbol{x}}^{SD} - w_{i0}\right)$ is called the *inner potential* of the unit. Parameters w_{ij} and w_{i0} are updated as detailed in the next section.

Layer 5 is a desensitization layer by action and has n neural units. The output of the i-th unit in this layer is calculated by

$$\bar{z}_i(\bar{y}_i) = \frac{1+\bar{y}_i}{2}\bar{z}_i.$$

Layer 6, which is the output layer, has single output unit z, which is determined from the number of units outputting $+1$ in layer 5. The output of layer 6 is calculated as follows:

$$\begin{aligned}z &= Q(\boldsymbol{x}, y) \\ &= g\left(\sum_{i\in I}\bar{z}_i(\bar{y}_i)\right),\end{aligned}$$

where I is index set $I = \{1,2,\ldots,n\}$ and $g(u)$ is a scaling function. z is also given by

$$z = g\left(\sum_{j\in J(y)}\bar{z}_j\right),$$

where $J(y) = \{j : \forall j \in I, \bar{y}_j = 1\}$.

3.2. SDNN-C Learning

SDNN-C learns based on error correction training.

When an agent moves to successive state s_{t+1} and receives reward r_{t+1}, it updates its action-value function by updating synaptic weights w_{ij} and threshold w_{i0}. This updating is done to minimize the distance between the current action value and a new action value given by Eq. (1).

Let z be the current action value and $\hat{z}$ be the new action values. Let u be the number of units outputting $+1$ in layer 5. Let $\hat{u}$ be the estimated number of units satisfying $\hat{z} = g(\hat{u})$. These variables determine the synaptic weights and thresholds to be updated and their updated values. The learning algorithm is detailed in **Fig. 2**.

4. Simulation Experiments

We conducted two simulation experiments to assess the effectiveness of our proposed SDNN-C. In these experiments, we used an acrobot swing-up task that differs in two ways from the original version [1]. Difference 1 is the continuity of state-action space. In these experiments, we used continuous state-action space. Difference 2 is the condition for completing the task. We set limits on the angular velocities for the two links and made it tighter as episodes elapsed. This condition gradually requires more precise action selection and contributes to evaluating the accuracy of the action-value function against continuous action. SDNN-C is used to approximate the action-value function for Q-learning. We quantified SDNN-C performance for the following points:

- The number of episodes required to learn a suitable action-value function

1. Divide a set of units not neutralized by the desensitization process in layer 4 into $|\hat{u}-u|$ subsets $Z_1, Z_2, \ldots, Z_{|\hat{u}-u|}$, preserving index order. Subsets should be of equal size, as far as possible.
2. If possible, select one unit in each set Z_i as follows: (a) if z is greater than $\hat{z}$, select the unit with the lowest magnitude of inner potential that outputs $+1$; (b) if z is less than $\hat{z}$, select the unit with the lowest magnitude of inner potential that outputs 0.
3. Update the synaptic weights and thresholds of selected units by Eqs. (2) and (3), respectively.

$$w_{ij} \leftarrow w_{ij} + c\,\mathrm{sgn}(\hat{u}-u)\,\bar{x}_j^{SD} \quad \ldots\ldots (2)$$

$$w_{i0} \leftarrow w_{i0} - c\,\mathrm{sgn}(\hat{u}-u), \quad \ldots\ldots (3)$$

where $\mathrm{sgn}(\cdot)$ is the sign function, and c is a learning coefficient.

Fig. 2. SDNN-C learning algorithm.

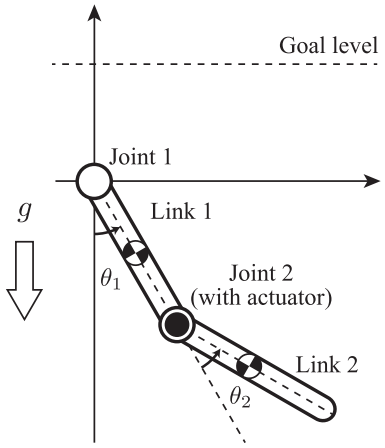


Fig. 3. Acrobot.

- The number of episodes until the number of steps per episode becomes larger again
- The number of steps in the final episode if an agent achieves a goal stably by the final episode

In experiment A, we confirm SDNN-C efficiency in continuous state-action space with redundant input. In experiment B, we confirm SDNN-C robustness in continuous state-action space with noisy and redundant input.

4.1. Acrobot Swing-Up Task

The *acrobot* [13] scheme is shown in **Fig. 3**. This underactuated two-link manipulator is commonly used as an RL benchmark in continuous state spaces [1, 5]. Joint 1 of the acrobot is freely movable and affixed to a wall. Joint 2 has an actuator that outputs the torque set by the controller. Torque τ is set from -10 Nm to $+10$ Nm. The agent selects one action in action set $\mathscr{A} = \{-10, -10+1/300, -10+2/300, \ldots, 0, \ldots, 10-1/300, 10\}$ given by equal divisions of the torque range. The state of the acrobot is described by angles θ_1, $\theta_2 \in [-\pi, \pi)$ and angular velocities $\dot{\theta}_1$, $\dot{\theta}_2$. We set physical parameters as follows: Link masses $m_1 = m_2 = 1$ kg; Link lengths $l_1 = l_2 = 1$ m; Distances to the center-of-mass of links $l_{c1} = l_{c2} = 0.5$ m; Moments of inertia of links $I_1 = I_2 = 1$ kg$\cdot$m^2; Gravitational acceleration $g = 9.8$ m/s^2.

The agent's goal in this task was to swing the tip of link 2 above the goal level in the minimum time and to momentarily stop it at the goal height. This task consists of subsequences called episodes. Each episode began with an initial state, $(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2) = (0, 0, 0, 0)$. The agent controlled the acrobot at 1-s intervals toward the goal. The agent reached the goal if the current state of the acrobot satisfied the following two conditions:

1. The tip reached 1.5 m above joint 1.
2. $|\dot{\theta}_1| < 3\pi\lambda$ and $|\dot{\theta}_2| < 5\pi\lambda$,

where variable λ specifies the difficulty in reaching the goal. λ is defined as

$$\lambda = \begin{cases} 10^{-\left(1+\frac{k-1}{10000-1}\right)} & (k \le 10000) \\ 10^{-2} & (k > 10000) \end{cases},$$

where k is the number of episodes. Note that reaching the goal becomes increasingly difficult as the number of episodes increases. If the acrobot attains its goal or fails to reach it within 50 s, it is restored to its initial state and a new episode is begun.

If the agent reaches its goal, it receives $+10$ as a reward. If $|\theta_1|$ is below 5° or angular velocities $\dot{\theta}_1$, $\dot{\theta}_2$ are outside of sensor range $[-3\pi, 3\pi]$, $[-5\pi, 5\pi]$, it receives -5. In all other states, it receives no reward whatsoever.

Initial outputs of function approximators were set to zero in all states and actions because we lacked previous knowledge of the acrobot completing this task.

4.2. Experiment A

In experiment A, we compare two cases: that with additional redundant inputs and that without.

4.2.1. Experimental Setup

The agent observed four normalized state variables $s_1, \ldots, s_4$ given by

$$s_1 = \frac{\theta_1}{2\pi} + 0.5,$$

$$s_2 = \frac{\theta_2}{2\pi} + 0.5,$$

$$s_3 = f_{\mathrm{sat}}\left(\frac{\dot{\theta}_1}{6\pi} + 0.5\right),$$

$$s_4 = f_{\mathrm{sat}}\left(\frac{\dot{\theta}_2}{10\pi} + 0.5\right),$$

$f_{\text{sat}}(x)$ is a saturation function defined as

$$f_{\text{sat}}(x) = \begin{cases} 1 & (1 \leq x) \\ x & (0 \leq x < 1) \\ 0 & (x < 0) \end{cases}.$$

Two types of redundant inputs, $s_5 \in [0,1]$ and $s_6 \in [0,1]$, were used. s_5 is the simplified kinetic energy of the acrobot and s_6 is the normalized height of the acrobot from joint 1. These were given by

$$s_5 = 2\Big((s_3 - 0.5)^2 + (s_4 - 0.5)^2\Big),$$
$$s_6 = \frac{1}{4}\Big(\cos(2\pi s_1) - \cos(2\pi(s_1 + s_2))\Big) + 0.5.$$

In this task, the agent was required to sequentially select correct actions. To prevent random actions from being selected during this evaluation, we specified two types of episodes—*learning episodes* and *test episodes*. During a learning episode, the agent determined action based on the ε-greedy policy and learned the action-value function. During the first 10 steps, probability ε was set to 0 and thereafter to 0.1. RL parameters were $\alpha = 0.5$ and $\gamma = 0.9$. After each interval of 10 learning episodes, one test episode was executed. During a test episode, the agent used the greedy policy without learning the action-value function. The test episode evaluated the agent's performance throughout previous learning episodes.

We conducted 10 trials and evaluated the number of steps per test episode in each case. Each trial consisted of 15,000 learning episodes and 1500 test episodes.

SDNN-C parameters were $m = 200$, $n = 900$, $n' = 300$, and $c = 0.1$, enabling SDNN-C to express 601 action values. The output range of SDNN-C was $[-20, 20]$, and the scaling function $g(u)$ in layer 6 was defined by

$$g(u) = \begin{cases} 20 & (280 < u) \\ \dfrac{2}{13}u - \dfrac{300}{13} & (20 \leq u \leq 280) \\ -20 & (u < 20) \end{cases}.$$

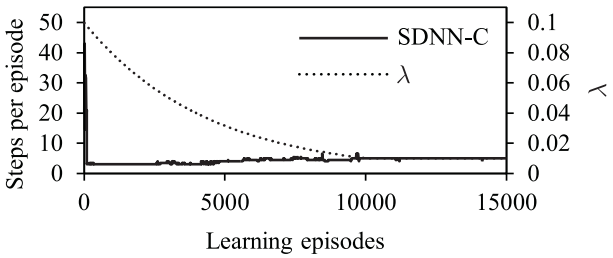
All synaptic weights were assigned a small random value. Even-numbered units were assigned large positive thresholds and odd-numbered large negative thresholds.

4.2.2. Experiments and Results

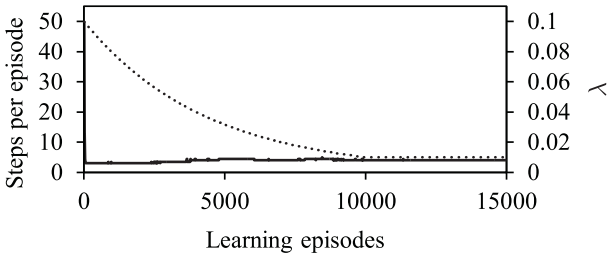
Experiment A results are shown in **Fig. 4**. The vertical axis indicates the number of steps in the test episode for each learning episode and the median number of steps among 10 trials is plotted.

These graphs show that the SDNN-C agent learned the action-value function within about 150 learning episodes, finally achieved the goal within 10 steps in both cases. This shows the robust performance of our proposal in redundant state space dimensions.

The computational cost of SDNN-C increased quadratically as the number of inputs increased, whereas the cost of a conventional method such as RBFN increased exponentially. This property is advantageous in the case of redundant input. In a case of 6-dimensional state space, for example, computation time for one step using SDNN-C and RBFN with 4 basis functions for each dimension was about 50 ms and 200 ms, respectively. Note that the higher the dimension, the greater the difference between RBFN and SDNN-C.



(a) The SDNN-C agent observing only 4 normalized state variables.



(b) The SDNN-C agent observing 4 normalized state variables plus 2 redundant variables.

Fig. 4. Experiment A results.

4.3. Experiment B

In experiment B, we conducted simulation experiments using an acrobot swing-up task with noisy sensor input to confirm SDNN-C robustness against noise and redundant input. We also used SDNN-C as a value function approximator for Q-learning. Although RBFN is well known as a function approximator in RL, it could not be used for this experiment due to its high computational cost. In this experiment, we used fuzzy Q-learning comparatively because its computational cost is low enough below RBFN on the parameters configured to make them perform at the same level approximation accuracy.

4.3.1. Fuzzy Q-Learning

For comparison, we used Jouffe's fuzzy Q-learning (FQL) algorithm [8]. The FQL agent has N rules activated for different states. Each rule R_i has parameter vector v^i representing the desirability of actions in discrete action set U.

The agent selects action a_t based on activated rules and its parameter vectors as follows:

$$a_t = \sum_{R_i \in A(s_t)} \varepsilon\text{-Greedy}(v^i) \cdot \alpha_{R_i},$$

$A(s_t)$ is the activated rule set in state s_t, α_{R_i} is the output of rule R_i (called a truth value) and ε-Greedy$(\cdot)$ is a function that selects a discrete action in U based on v^i. In

Table 1. Cases of redundant state space.

Name of case	Components of state observed by the agent
Case I	(s_1, s_2, s_3, s_4)
Case II	$(\tilde{s_1}, \tilde{s_2}, \tilde{s_3}, \tilde{s_4})$
Case III	$(\tilde{s_1}, \tilde{s_2}, \tilde{s_3}, \tilde{s_4}, \tilde{s_5})$
Case IV	$(\tilde{s_1}, \tilde{s_2}, \tilde{s_3}, \tilde{s_4}, \tilde{s_5}, \tilde{s_6})$

his original study, Jouffe proposed and used another type of ε-greedy function, but in this experiment, we used the same function applied to SDNN-C.

Parameter updating is based on Eq. (1)

$$\forall R_i \in A(s_t),$$
$$v^i[a_t^i] \leftarrow v^i[a_t^i] + c\left(r_{t+1} + \gamma Q_t^*(s_{t+1}) - Q_t(s_t, a_t)\right)\alpha_{R_i},$$

$v^i[a_t^i]$ is a parameter for the action selected by rule R_i at time t, c is a learning coefficient and $Q^*(s)$ represents the optimal action value in state s. $Q_t^*(s_{t+1})$ and $Q_t(s_t, a_t)$ are calculated by

$$Q_t^*(s_{t+1}) = \sum_{R_i \in A(s_{t+1})} \left(\max_{a \in U} v^i[a]\right)\alpha_{R_i},$$
$$Q_t(s_t, a_t) = \sum_{R_i \in A(s_t)} v^i[a_t^i]\alpha_{R_i}.$$

4.3.2. Experimental Setup

Experiment B was conducted for the four cases in **Table 1**. Case I contained no redundant input or sensor noise.

In contrast, in case II, state variables observed by the agent included independent Gaussian noise, which were calculated as follows:

$$\tilde{s_1} = f_{\text{wrap}}(s_1 + e_1),$$
$$\tilde{s_2} = f_{\text{wrap}}(s_2 + e_2),$$
$$\tilde{s_3} = f_{\text{sat}}(s_3 + e_3),$$
$$\tilde{s_4} = f_{\text{sat}}(s_4 + e_4),$$

$e_i (i = 1, \ldots, 4)$ is Gaussian noise (mean $\mu = 0$ and standard deviation $\sigma = 0.02$) and $f_{\text{wrap}}(x)$ is a function that wraps x to the interval $[0, 1]$.

In cases III and IV, the agent also observed noisy and redundant inputs $\tilde{s_5}$ and $\tilde{s_6}$, given by

$$\tilde{s_5} = f_{\text{sat}}(s_5 + e_5),$$
$$\tilde{s_6} = f_{\text{sat}}(s_6 + e_6).$$

FQL parameters were determined by preliminary experiments. Specifically, each state dimension was partitioned at even intervals with nine triangular membership functions. The number of fuzzy rules N was 9^d, where d is the total number of state dimensions. Discrete action sets U of these rules were the same for all rules, and contained 21 actions given by equally dividing the torque range.

Trial settings and the parameter of SDNN-C and Q-learning were the same as those used in experiment A.

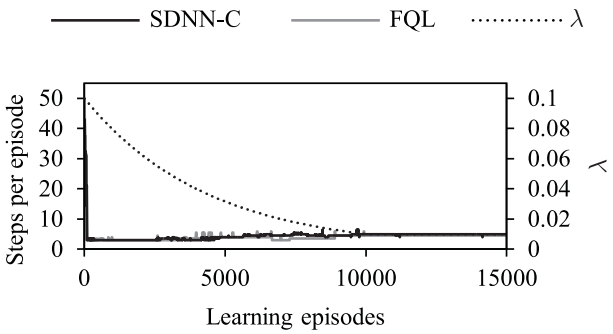
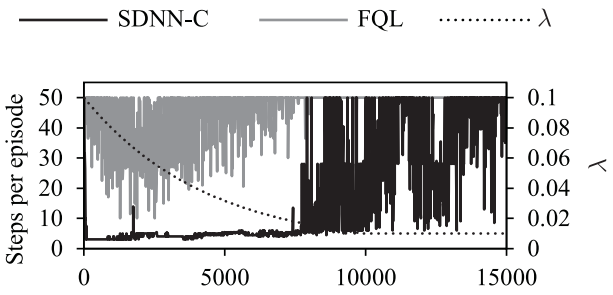
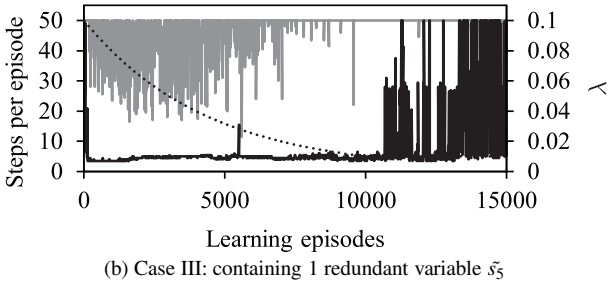


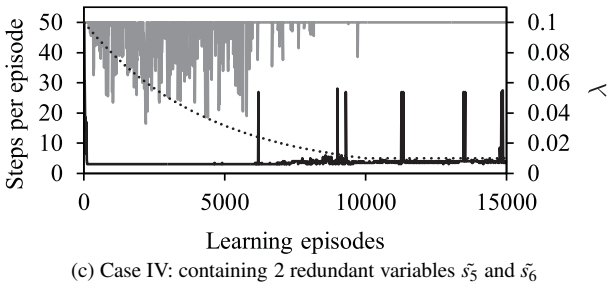
Fig. 5. Experiment B results for Case I.



(a) Case II: containing no redundant variable



(b) Case III: containing 1 redundant variable $\tilde{s_5}$



(c) Case IV: containing 2 redundant variables $\tilde{s_5}$ and $\tilde{s_6}$

Fig. 6. Experiment B results for Cases II–IV.

4.3.3. Experiments and Results

Experiment B results are presented in **Fig. 5**, representing case I, and **Fig. 6**, representing cases II–IV, where the median number of steps with SDNN-C and FQL among the 10 trials is plotted.

We compared the influence of sensor noise on both methods in **Figs. 5** and **6(a)**. These graphs show that the performance of the FQL agent depended strongly on the presence or absence of sensor noise. Although the FQL agent could learn within approximately 100 episodes in a noise-free case, it failed to learn stable upswing actions even in the early episodes in the noisy case. In case

II, SDNN-C agent performance in early episodes was almost equal to both methods in case I. After that, it began worsening at about episode 7,500, and the agent failed to achieve the goal stably due to the relative increase in the noise influence.

Figure 6 shows that the FQL agent failed to learn stable upswing actions due to the effect of sensor noise in all cases. In contrast to FQL in early episodes, the SDNNC agent learned the suitable action-value function within almost the same number of episodes regardless of additional redundant input. The SDNN-C agent in case III kept a suitable action selection until a later episode than in case II. In particular, in case IV, the SDNN-C agent finally achieved the goal in about 4 steps, which is almost the same as in case I. Although SDNN-C performance may depend on the type of additional input, noise tolerance tended to improve as the number of redundant inputs increased.

Based on these results, we confirmed that SDNN-C is strongly robust against sensor noise and redundant input and takes better actions by using information from additional inputs. We consider that these properties are realized thanks to powerful, nonlocal generalization and the high representation capability of SDNN, by which, designing state space may become easier as system designers are enabled to add any input that may be related to control.

5. Conclusions

We have proposed SDNN-C, novel value-function approximation for Q-learning in continuous state-action space based on SDNN. We evaluated this method in an acrobot task extended to continuous state-action space including noisy and redundant input. In experiment A, we confirmed that SDNN-C is applicable to continuous state-action space with redundant input. Experiment B results demonstrated that SDNN-C is strongly robust against sensor noise and redundant input, thus enabling an agent to take better action by using additional input without learning efficiency deteriorating. We consider that these properties have been realized by the powerful, nonlocal generalization and the high representation capability of SDNN. These are advantageous in real-world applications such as robotic systems.

In future studies, we plan to apply our proposal to extend methods to multidimensional action space. We also plan to experimentally compare the performance of our proposed method to that of other RL algorithms such as actor–critic methods.

Acknowledgements

This study was supported partly by JSPS KAKENHI Grant Numbers 22300079, 24760308, and 26590173.

References:

[1] R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction," MIT Press, 1998.
[2] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. thesis, University of Cambridge, 1989.
[3] C. J. C. H. Watkins and P. Dayan, "Q-learning," Machine Learning, Vol.8, pp. 279-292, 1992.
[4] C. Gaskett, D. Wettergreen, and A. Zelinsky, "Q-learning in continuous state and action spaces," Proc. 12th Australian Joint Conf. on Artificial Intell., Sydney, pp. 417-428, 1999.
[5] G. Konidaris, S. Osentoski, and P. Thomas, "Value function approximation in reinforcement learning using the Fourier basis," Proc. 25th Conf. on Artificial Intell., San Francisco, pp. 380-385, 2011.
[6] A. Geramifard, M. Bowling, and R. S. Sutton, "Incremental least-square temporal difference learning," Proc. 21th Conf. on Artificial Intell., Boston, pp. 356-361, 2006.
[7] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," Neural Computation, Vol.3, No.2, pp. 246-257, 1991.
[8] L. Jouffe, "Fuzzy inference system learning by reinforcement methods," IEEE Trans. on Systems, Man, and Cybernetics, Part C: Applications and Reviews, Vol.28, No.3, pp. 338-355, 1998.
[9] P. Y. Glorennec, "Reinforcement learning: An overview," Proc. European Symposium on Intelligent Techniques (ESIT-00), Aachen, pp. 14-35, 2000.
[10] K. Nonaka, F. Tanaka, and M. Morita, "Empirical comparison of feedforward neural networks on two-variable function approximation," IEICE Trans. Inf. & Syst.(Japanese Edition), Vol.94, No.12, pp. 2114-2125, 2011.
[11] K. Horie, A. Suemitsu, and M. Morita, "Direct estimation of hand motion speed from surface electromyograms using a selective desensitization neural network," J. Signal Process., Vol.18, No.4, pp. 225-228, 2014.
[12] T. Kobayashi, T. Shibuya, and M. Morita, "Q-learning in Continuous State-Action Space with Redundant Dimensions by Using a Selective Desensitization Neural Network," Proc. SCIS&ISIS 2014, Kitakyushu, pp. 801-806, 2014.
[13] M.W. Spong, "The swing up control problem for the acrobot," IEEE Control Syst. Mag., Vol.15, No.1, pp. 49-55, 1995.

Name:
Takaaki Kobayashi

Affiliation:
Faculty of Engineering, Information and Systems, University of Tsukuba

Address:
1-1-1 Tennodai, Tsukuba, Ibaraki, 305-8573 Japan

Brief Biographical History:

2006- College of Engineering Systems, University of Tsukuba
2010- Department of Intelligent Interaction Technologies, University of Tsukuba
2015- Faculty of Engineering, Information and Systems, University of Tsukuba

Main Works:

• Neural networks and reinforcement learning
• "Q-Learning in Continuous State-Action Space by Using a Selective Desensitization Neural Network," IEICE Trans. Inf. & Syst. (Japanese Edition), Vol.98, No.2, pp. 287-299, 2015.

Name:
Takeshi Shibuya

Affiliation:
Faculty of Engineering, Information and Systems, University of Tsukuba

Address:
1-1-1 Tennodai, Tsukuba, Ibaraki, 305-8573 Japan

Brief Biographical History:
2007-2010 Ph.D. Candidate, Graduate School of Engineering, Yokohama National University
2007-2010 Research Fellow, the Japan Society for the Promotion of Science
2011- Assistant Professor, University of Tsukuba

Main Works:
• "Complex-Valued Reinforcement Learning: a Context-Based Approach for POMDPs," Advances in Reinforcement Learning, pp. 255-274, InTech, 2011.

Membership in Academic Societies:
• The Institute of Electrical and Electronics Engineers (IEEE)
• Institute of Electronics, Information and Communication Engineers (IEICE)
• The Institute of Electrical Engineers of Japan (IEEJ)

Name:
Masahiko Morita

Affiliation:
Faculty of Engineering, Information and Systems, University of Tsukuba

Address:
1-1-1 Tennodai, Tsukuba, Ibaraki, 305-8573 Japan

Brief Biographical History:
1991 Received Doctoral degree of Engineering from the University of Tokyo
1992.6-2000.3 Lecturer, University of Tsukuba
2000.3-2007.3 Assistant Professor, University of Tsukuba
2007.4-2007.5 Associate Professor, University of Tsukuba
2007.6- Professor, University of Tsukuba

Main Works:
• Biological information processing and neural computation
• "Associative memory with nonmonotone dynamics," Neural networks, Vol.6, No.1, pp. 115-126, 1993.

Membership in Academic Societies:
• The Society of Instrument and Control Engineers (SICE)
• The Institute of Electronics, Information and Communication Engineers (IEICE)
• Japanese Neural Network Society (JNNS)