STATE GENERALIZATION WITH SUPPORT VECTOR MACHINES IN REINFORCEMENT LEARNING

Ryo Goto, Toshihiro Matsui and Hiroshi Matsuo

Department of Electrical and Computer Engineering, Nagoya Institute of Technology
Gokiso-cho, Showa-ku, Nagoya, 466-8555, JAPAN
{rgoto,tmatsui}@mars.elcom.nitech.ac.jp, matsuo@elcom.nitech.ac.jp

ABSTRACT

The conventional reinforcement learning assumes discrete state space. Therefore, it is necessary to make states discrete manually in order to handle continuous state environments. However, if the simple discretization is applied, the number of states increases exponentially with the dimension of the state space, and the learning time increase.

In this paper, we propose a state generalization that is able to quickly adapt to environments by using Support Vector Machines. To evaluate this method, we do an experiment on the simulation task that navigates a robot to a goal.

1. INTRODUCTION

Recent advancement of the robot technology raised the necessity of the learning robot that autonomously adapts to environment. As an effective technique to learn effective control rules, reinforcement learning has attracted many attentions. Reinforcement learning is framework of the learning which automatically acquires control rules from the given reward signal.

The conventional reinforcement learning methods are targeting discrete state spaces. Therefore, when continuous states should be handled, it is necessary to discretize the state space appropriately. However, in the high dimensional state space, discretization causes the increase in the number of states, and the learning time and required memories increase remarkably. This problem is serious when we consider applying reinforcement learning to a real task because a state input is often given as a continuous multidimensional vector.

To address this problem, there is an approach of approximating the value function with the continuous function[1][2]. In this approach, it is possible to choose effective action even for inexperienced states by exploiting old experiences, but a lot of trial is needed to adapt to complicated environment. There is also another approach that generalizes states and constructs the state space appropriately. The methods adopting this approach are also proposed[3], however most of them use simple models.

In this paper, we propose a method that learns effectively with a few experience and quickly adapts to environment by generalizing multidimensional continuous states using Support Vector Machines (SVMs)[4]. SVM is a two-class pattern recognition method with some advantages. Our method is able to generalize more flexibly than conventional techniques.

In the subsequent sections, we provide overview of reinforcement learning and SVMs. Then, we describe the proposed method and do an experiment on the simulation task that navigates a robot to a goal to evaluate this method.

2. REINFORCEMENT LEARNING

Reinforcement learning is one of the non-supervised learning technique in which the reward signal is given not for an individual action but for a series of actions.

Typical reinforcement learning algorithms such as Q-learning[5] evaluate state values based on the rewards. The state value is the total expected discounted reward attained by the optimal policy starting from the state. The method handling continuous state space by approximating the value function with the continuous function such as Incremental NGnet (INGnet) [2] is proposed. INGnet is a method that extends radial basis function networks and learns the value function for the continuous state space by placing basis functions if needed.

3. SUPPORT VECTOR MACHINE

Support Vector Machine (SVM) is a two-class pattern recognition method and learns the decision function $f(\boldsymbol{x})$ for determining the class of input $\boldsymbol{x}$ from N training data $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$, where $\boldsymbol{x}_i \in \mathbf{R}^n$ is a

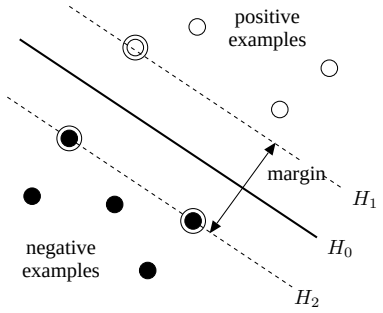

Figure 1: The separation of linear SVM

training example and $y_i \in \{-1, 1\}$ is a class assigned to $\boldsymbol{x}_i$.

3.1. Linear SVM

Figure 1 shows the concept of separation of linear SVM. As shown in this figure, we suppose that all training data are separable linearly. $H_0 : \boldsymbol{w} \cdot \boldsymbol{x} + b = 0$ is a hyperplane which separate positive examples and negative examples, where $\boldsymbol{w}$ is the normal vector of this hyperplane and b is a constant. The nearest positive example to H_0 and the nearest negative example to H_0 are on $H_1 : \boldsymbol{w} \cdot \boldsymbol{x} + b = 1$ and $H_2 : \boldsymbol{w} \cdot \boldsymbol{x} + b = -1$ parallel to H_0, respectively. The distance between H_1 and H_2 is called margin. SVM is an algorithm for finding the separation hyperplane H_0 that maximize the margin.

No positive examples exist in H_0 side from H_1 and no negative examples exist in H_0 side from H_2, so we have the following equation:

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x}_i + b) - 1 \geq 0 \tag{1}$$

To maximize the margin, we should maximize $1/\|\boldsymbol{w}\|$. This can be formulated as the problem that minimizing $\frac{1}{2}\|\boldsymbol{w}\|^2$ subject to Equation 1. This problem can be transformed into a dual quadratic programming problem using Lagrange multipliers $\boldsymbol{\alpha} = \{\alpha_1, \ldots, \alpha_l\}$. When $\boldsymbol{\alpha}$ is solved, we can get $\boldsymbol{w} = \sum_{i=1}^{l} \alpha_i y_i \boldsymbol{x}_i$, and the decision function is

$$f(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + b = \sum_{i=1}^{l} \alpha_i y_i \boldsymbol{x}_i \cdot \boldsymbol{x} + b \tag{2}$$

We provided overview of the perfect linear separation, however this is applicable in the case of imperfect separation.

Instead of solving standard quadratic programming problem, we can solve $\boldsymbol{\alpha}$ by using Sequential Minimal Optimization (SMO) algorithm[6]. We used this algorithm in this research.

3.2. Nonlinear SVM

In nonlinear SVM, training data is mapped to the other high dimensional space $\mathcal{H}$ by $\Phi : \mathbf{R}^n \mapsto \mathcal{H}$, and separated nonlinearly by applying linear separation on $\mathcal{H}$. In this case, the decision function is

$$f(\boldsymbol{x}) = \sum_{i=1}^{l} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b \tag{3}$$

where $K(\boldsymbol{x}, \boldsymbol{y}) = \Phi(\boldsymbol{x}) \cdot \Phi(\boldsymbol{y})$. Although it is difficult to find mapping Φ, it is not necessary to know Φ if inner products on $\mathcal{H}$ can be calculated. The function $K(\boldsymbol{x}, \boldsymbol{y})$ is called kernel function and some valid kernel functions are known. In this research, we use

$$K(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\frac{\|\boldsymbol{x} - \boldsymbol{y}\|^2}{2\sigma^2}) \tag{4}$$

where σ is a constant.

4. STATE GENERALIZATION USING SVM

It is necessary to generalize multiple states as a state because states exist innumerably in the continuous state space. However, many of conventional methods use the generalization based on a simple model like an ellipsoidal model[3]. Therefore, construction of state space will be complicated in some environments and estimation for unknown states is weak.

In this section, we propose the state generalization method in which the state value is estimated and the states where value rises after action are generalized using SVM. SVM is a technique that is able to separate data nonlinearly, and it can generalize states in multidimensional continuous state space more flexibly than conventional methods. Therefore, the proposed method can presume the optimal action at inexperienced states by exploiting the function of SVM and adapts to environment from a few experience. Below, we describe each procedure of the proposed method.

4.1. Collection of state transitions

To generalize states, it is necessary to get states and their attributes. Therefore, the state transitions are collected from N trials and each transition is expressed with $(\boldsymbol{s}, a, \boldsymbol{s}')$, where $\boldsymbol{s}$ is a state before action, a is an action and $\boldsymbol{s}'$ is a state after action. If we apply SVM to all examples, the computation time will increase because of the large number of examples. So we prepared the following restrictions:

- Only the examples of L steps backward from the step that reached the goal state are collected.

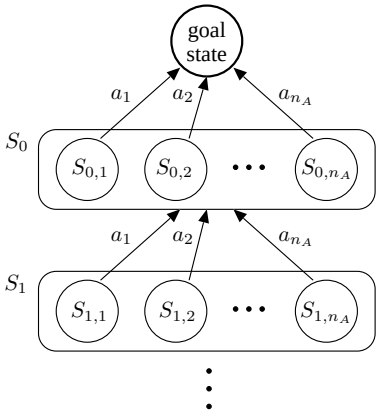

Figure 2: Estimation of state value

- If the action of the new transition is identical to the action of already collected transition and both of the distance between the states before action and the distance between the states after action are smaller than threshold d_{th}, then the new state transition is not collected.

4.2. Estimation of state value

The proposed method finally decides whether the state value rises as a result of performing a certain action. To do this, the state value must be known. In the environment in which the reward is placed only at the goal state, state value is considered to be the number of optimal steps to the goal state. In this method, the state space is constructed through operations that generalize the states that reach the goal state or the already generalized state at one step, as shown in Fig. 2. The algorithm is stated as follows.

1. Let D_k be a set of transitions in D whose action is $a_k \in A$ $(k = 1, \ldots, n_A)$, where D is a set of collected transitions, A is a set of performable actions and n_A is the number of elements in A.

   For each transition $\delta_i = (\boldsymbol{s}_i, a_k, \boldsymbol{s}'_i) \in D_k$, $\boldsymbol{s}_i$ is labeled as the positive example if $\boldsymbol{s}'_i$ is the goal state. Otherwise, $\boldsymbol{s}_i$ is labeled as the negative example. And they are separated by SVM.

2. Let $f_{0,k}$ be the decision function computed in procedure 1 and let $S_{0,k} = \{\boldsymbol{s} | f_{0,k}(\boldsymbol{s}) > 0\}$. Procedure 1 is performed for all $k = 1, \ldots, n_A$, thus union set $S_0 = \bigcup_{k=1}^{n_A} S_{0,k}$ is constructed.

3. For each transition $\delta_i = (\boldsymbol{s}_i, a_k, \boldsymbol{s}'_i) \in D_k$, $\boldsymbol{s}_i$ is newly labeled as the positive example if $\boldsymbol{s}'_i \in S_0$. Otherwise, $\boldsymbol{s}_i$ is labeled as the negative example. However, if $\boldsymbol{s}_i$ is labeled as a positive example before this procedure, it is not labeled. Then, they are separated by SVM and $S_1 = \bigcup_{k=1}^{n_A} S_{1,k}$ is constructed like procedure 2.

4. Similarly, this algorithm constructs sets from S_0 to S_{m-1}, where m is the number of estimating state values.

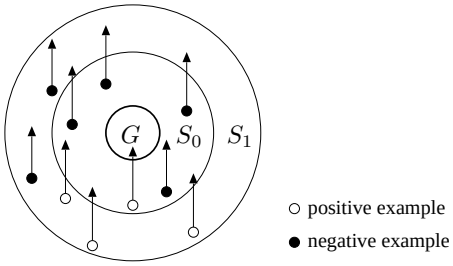

Figure 3: An example of the attributes of states

4.3. State generalization

Because the states that have same values were only generalized in previous subsection, state value after an action can not be presumed globally. In the proposed method, the states where the values rise after an action are generalized using SVM for each action, and whole state space is constructed from two classes. By this generalization, it is possible to choose the good action even at the inexperienced state.

1. $\delta_i = (\boldsymbol{s}_i, a_k, \boldsymbol{s}'_i) \in D_k$ is a state transition in the collected transitions whose action is a_k. Let S_v be a set of the highest value among the sets that include $\boldsymbol{s}_i$, and $S_{v'}$ be a set of the highest value among the sets that include $\boldsymbol{s}'_i$. $\boldsymbol{s}_i$ is collected as a positive example when $v > v'$ or when S_v does not exist and $S_{v'}$ exists. And it is collected as a negative example when $v \leq v'$ or when S_v exists and $S_{v'}$ does not exist. Figure 3 shows an example of the attributes of states in state generalization.

2. After applying procedure 1 to each transition in D_k, the states are generalized by SVM.

3. Procedure 1 and 2 are applied to all $a_k \in A$.

4.4. Selecting action

The generalization error affects the performance of this method. Therefore, this method selects the action at random by probability ϵ, and selects the action that classifies the input into a positive class by probability

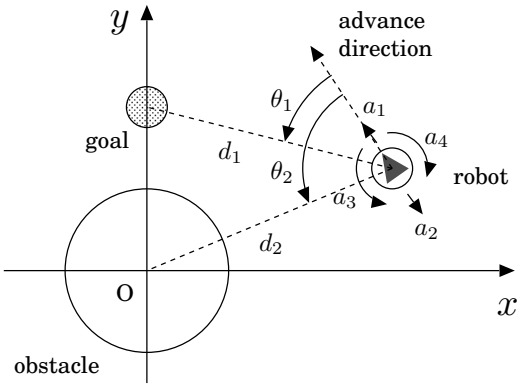

Figure 4: The task to assume

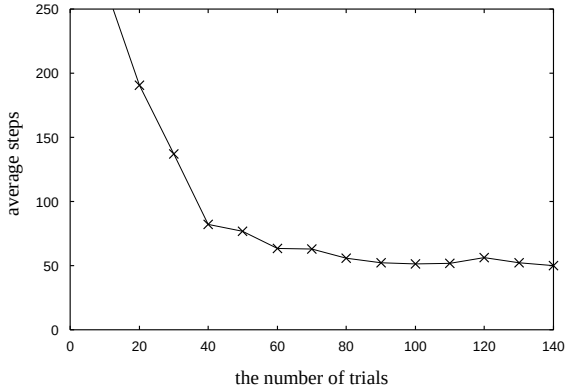

Figure 5: The average number of steps in the proposed method

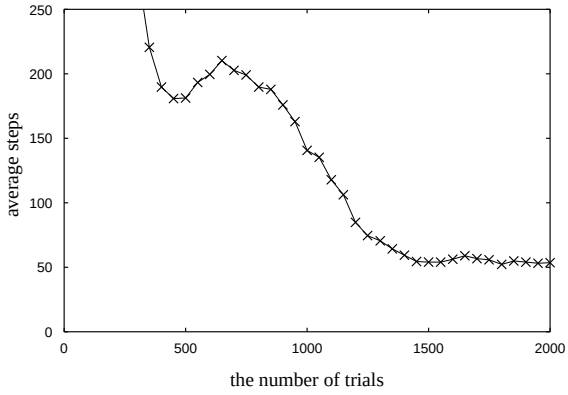

Figure 6: The average number of steps in INGnet

$1 - \epsilon$. However, if the input is classified into different action's positive classes, one of them is adopted at random, and if the input cannot be classified into any action's positive class, an action is chosen at random out of all actions.

5. EXPERIMENTS

5.1. The task to assume

As shown in Fig. 4, we did an experiment on the simulation task that navigates a robot to a goal. The goal is a circle with center $(x, y) = (0.0, 0.8)$ and radius 0.1, and the obstacle is a circle with center $(x, y) = (0.0, 0.0)$ and radius 0.4. The initial location of the robot is $x = [-0.5, 0.5]$, $y = -0.7$. The reward is generated only when the robot reaches the goal. The valid range is $-2.0 \leq x \leq 2.0$ and $-2.0 \leq y \leq 2.0$, and the robot is returned to an initial location when it goes out of valid range.

Actions of robot are a_1 (1.0 advance), a_2 (1.0 retreat), a_3 (0.2π[rad] rotations of the advance direction) and a_4 (-0.2π[rad] rotations of the advance direction). When the robot tries to perform action that collides with the obstacle, the action does not perform. However, a_3 and a_4 are always performed because the robot is regarded as a point.

d_1 is the distance between the robot and the center of goal and θ_1[rad] ($-\pi < \theta_1 \leq \pi$) is the relative angle to the center of goal from the robot's advance direction. Similarly, d_2 and θ_2[rad] ($-\pi < \theta_2 \leq \pi$) are the distance and the relative angle between the robot and the center of obstacle. The robot observes 4-dimensional state $(d_1, \theta_1, d_2, \theta_2)$.

5.2. Parameter settings

The maximum number of examples collected in each trial is $L = 300$, the number of estimated state values is $m = 30$, the threshold of distance between two states is $d_{th} = 0.2$, and the probability in the action selection is $\epsilon = 0.3$. And for the parameter σ of kernel function, $\sigma = 0.3$ in estimation of state value, and $\sigma = 0.5$ in state generalization. The reason why the σ value for state generalization is bigger is to constitute state space more globally.

5.3. Comparison with INGnet

We did an experiment on this simulation with the proposed method and INGnet.

We measured the amount of memories that is needed for learning through 10 experiments. On average, the proposed method required 4476 vectors for state gen-

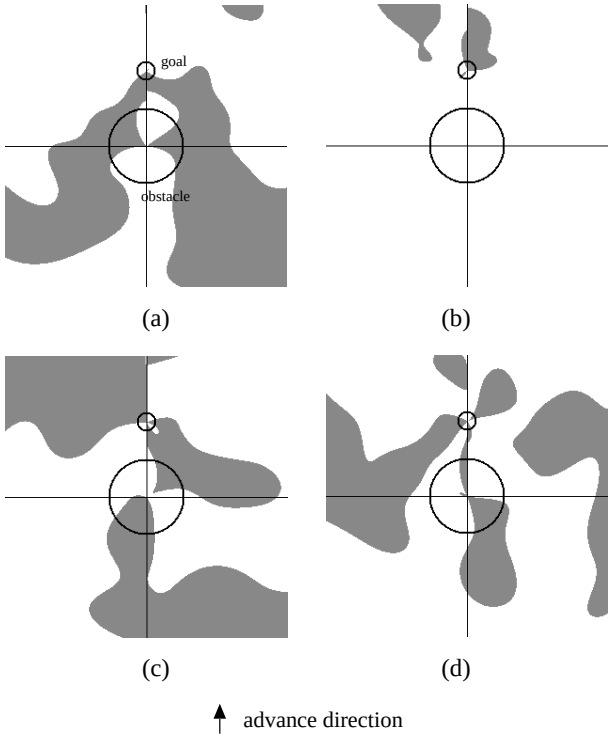

Figure 7: The example of construction of the state space in each action: (a) a_1 (advance), (b) a_2 (retreat), (c) a_3 (left rotation), (d) a_4 (right rotation)

eralization and INGnet required 988 bases. The reason why the proposed method required more memories than INGnet is that the proposed method needs to construct the state space for every action.

The result of this simulation using the proposed method and INGnet are shown in Fig. 5 and Fig. 6, respectively. The value of y-axis is the average of the number of steps taken to reach the goal state in 1000 simulations, and the agent relearned after changing the random seed every 100 simulations. As shown in Fig. 5 and Fig. 6, the number of steps converged to near 50 steps in both methods. However, while it was converged by the 1500th trial in INGnet, it was converged by the 80th trial in the proposed method. Thus, this method can adapt to environment faster than INGnet.

5.4. Construction of the state space

Figure 7 shows the example of construction of the state space after 140 learning trials in the proposed method. The gray region of Figure 7(a) expresses the set of states in which it is expected that the state value rises after performing action a_1 when the robot's advance direction is north ($(x, y) = (0, 1)$). Similarly, Figure 7(b), (c) and (d) express the construction of the state space in action a_2, a_3 and a_4, respectively. In this example, the states classified into the wrong class are seen, and we consider this was caused by the generalization error of SVM and reduction of training examples. However, there is no bad influence of error so much since selection of action is stochastic.

6. CONCLUSION

In this paper, we proposed a state generalization method that is able to quickly adapt to environments by Support Vector Machines. And by the experiment, we showed that our method quickly adapted to the environment compared with Incremental NGnet. In the future work, we will examine how to reduce the misclassification, and will compare with the other generalization methods.

7. REFERENCES

[1] J. A. Boyan and A. W. Moore, Generalization in Reinforcement Learning: Safely Approximating the Value Function, in G. Tesauro, D. S. Touretzky, and T. K. Leen eds., *Advances in Neural Information Processing Systems 7*, pp.369–376. MIT Press, 1995.

[2] J. Morimoto and K. Doya, Learning Dynamic Motor Sequence in High-dimensional State Space by Reinforcement Learning: Learning to Stand Up, *IEICE Transactions on Information and Systems*, Vol.J82-D2, No.11, pp.2118–2131, 1999.

[3] M. Asada, S. Noda, and K. Hosoda, Action-Based Sensor Space Categorization for Robot Learning, in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems 1996 (IROS '96)*, pp.1502–1509, 1996.

[4] B. E. Boser, I. M. Guyon, and V. N. Vapnik, A Training Algorithm for Optimal Margin Classifiers, in *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pp.144–152, 1992.

[5] C. J. C. H. Watkins and P. Dayan, Technical Note: Q-Learning, *Machine Learning*, Vol.8, No.3, pp.279–292, 1992.

[6] J. C. Platt, Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines, *Technical Report MSR-TR-98-14*, Microsoft Research, 1998.