Behavior Learning of Autonomous Robots by Modified Learning Vector Quantization[†]

Shon MIN KYU*, Junichi MURATA* and Kotaro HIRASAWA*

This paper presents a method for searching for the optimal paths for autonomously moving agents in mazes by modified Learning Vector Quantization(LVQ) in a reinforcement learning framework. LVQ algorithm is faster than Q-learning algorithms because LVQ concentrates on the best behavior in available behaviors while Q-learning algorithms calculate values of all available behaviors and choose the best behavior among them. However, ordinary LVQ sometimes mis-learns in the reinforcement learning environment due to erroneous teacher signals. Here a new LVQ algorithm is proposed to overcome this problem, which finds the optimal path more efficiently.

Key Words: robot path finding, Learning Vector Quantization(LVQ), reinforcement learning

1. Introduction

Reinforcement learning¹⁾ is a technique to learn how to act in an unknown environment through trial and error without specifying how the task is achieved, and has been widely used in behavior learning of autonomous agents in unknown environments, e.g. optimal path finding in a maze. It requires only a scalar reinforcement signal, reward or punishment, as the performance feedback from the environment.

Q-learning²⁾ is one of the most famous algorithms for the reinforcement learning. A learning agent estimates the value called Q-value. Q-value is the evaluation for the state and action pair of the agent, and the learning agent selects an action based on the Q-value. The original Q-learning algorithm refers to the table, called Q-table, which is prepared for all pairs of discrete states and discrete actions. Q-learning has an advantage that the learning result is correct but also has a disadvantage that the learning time is long.

This paper presents a method for searching for the optimal paths for autonomously moving agents by modified Learning Vector Quantization (LVQ) in a reinforcement learning framework.

Kohonen's LVQ^{3)~5)} algorithm is faster when used in a reinforcement learning framework than Q-learning algorithm. The Q-learning algorithm calculates Q-values

of all available actions for input information(state) obtained from the environment and chooses the action with the highest Q-value among them. On the other hand, the LVQ network is basically a system for pattern classification and thus maps the input information directly to the best action.

The LVQ networks have weight vectors or prototypes each of which has a label (output value of output node). They are trained by supervised learning algorithms where pairs of sample input vector and its associated correct label are given as training data.

In the reinforcement learning framework, the labels correspond to the actions. However, during the training process, it cannot be known which is the correct action. Only available is whether an action has got a reward from the environment or not. One possible way is to regard an action as the correct action for an input vector if that action has got a reward. For example, in an optimal path finding problem, an action 'go-right' may eventually leads to the goal and thus may get a reward. However another action 'go-forward' might have been the best action in the sense that it takes the agent to the goal most quickly. Here, the reward is erroneous information for the supervised learning. A new LVQ algorithm must be devised which can deal with these erroneous and incomplete training data in the reinforcement learning framework.

The paper is organized as follows. First, Kohonen's LVQ algorithm and the possible difficulties will be investigated in Section 2 which may occur when it is used in the reinforcement learning framework. In Section 3, a new LVQ algorithm is proposed to overcome the problem described in Section 2. Section 4 gives examples of application of proposed algorithm. Section 5 contains concluding

[†] presented on the 9th FAN Symposium (1999 · 10)

* Graduate School of Information Science and Electrical Engineering, Kyushu University, 6-10-1 Hakozaki, Higasi-ku, Fukuoka

(Received March 2, 2001)

(Revised August 22, 2001)

remarks.

2. LVQ Algorithm

2.1 LVQ algorithm

Kohonen's LVQ is a supervised learning algorithm associated with the competitive network shown in Fig.1. The network consists of an input layer and an output layer. Each node in the input layer is connected directly to the output layer. A weight vector is associated with each node in the output layer as shown in Fig.1. Each weight vector has a label (output value of output node). The weights are trained by a supervised learning algorithm where pairs of sample input vector and its associated correct label are given as training data. The aim of training is to obtain appropriate weights that give correct labels for the input data.

Fig. 1 Structure of LVQ network

The LVQ algorithm can be summarized as follows:

- 1) Initialize the weight vectors w_i , $i=1, \dots, c$ of the LVQ network, where c is the total number of the output nodes.
- 2) Input the vector $x=[x_1, x_2, \dots, x_n]^T$ to the LVQ network.
- 3) Calculate distance d_j between the input vector x and weight vector w_j as in Eq.(1)

$$d_j = \sqrt{\sum_{i=1}^n (x_i - w_{ji})^2}, \quad (1)$$

where w_{ji} is weight between the i th node of input layer and the j th node of output layer.

- 4) Find the minimum among $d_j, j=1, \dots, c$ and denote it by d_l .
- 5) Update w_l as in Eq.(2)

$$w_l := w_l + \Delta w_l, \quad \Delta w_l = \begin{cases} +\eta(x - w_l) & : \text{if the selected output node } l \text{ is correct,} \\ -\eta(x - w_l) & : \text{otherwise.} \end{cases} \quad (2)$$

where $\eta > 0$ is the learning rate.

- 6) Go to step 2.

In Eq.(2), the LVQ network compares the label of selected output node l with output value of training data. If the selected output node l (the 'winner') is correct, the weight vector w_l of the selected output node l is updated so that it approaches the input vector x .

In the case of the ordinary LVQ algorithm, for a given training pair, its corresponding winner weight vector approaches the sample input vector, and if the same training pair is given repeatedly, the weight will approach the sample input vector without limit of distance.

After the training, the LVQ network chooses the nearest weight vector to a given input vector, and outputs the 'label' corresponding to the weight vector as the network output.

2.2 Difficulties of LVQ algorithm in the reinforcement learning

The weight vectors of LVQ network are trained by supervised learning algorithm. However the reinforcement learning is unsupervised learning. Thus the LVQ network cannot receive correct output vector or optimal action in the reinforcement learning framework. In the reinforcement learning framework, an output node corresponds to an action, and the learning agent selects the action associated with the winner node for a given input. However, during the training process, it cannot be known which is the correct action. The LVQ network learns only the action which gets the reward.

Let us consider again the optimal path finding example given in section 1. There are many paths from start to goal in a maze. All of actions which lead the learning agent to the goal can get a reward. An action 'go-right' may eventually leads to the goal and thus may get a reward. Therefore, the weight vector corresponding to 'go-right' may be updated to approach the current input vector by the LVQ algorithm. However another action 'go-forward' might have been the best action in the sense that it takes the agent to the goal most quickly. Once the 'go-right' is selected and its weight vector is update to approach the input vector, the same action becomes more likely to be selected. Therefore, the action 'go-right' will be repeatedly selected, and the weight vector associated with this sub-optimal action can be closer to the input vector than the optimal action 'go-forward' (See Fig.2). After learning, the learning agent may learn not the optimal action but the sub-optimal action

A new LVQ algorithm must be devised which can deal with these erroneous and incomplete training data in the reinforcement learning framework.

Fig. 2 Difficulties of training of weight vectors associated with sub-optimal action and optimal action

Fig. 3 Movement of weight vectors by modified LVQ Algorithm

3. Modified LVQ Algorithm

Here a new LVQ algorithm is proposed to overcome the above problem, which finds the optimal weight vector more efficiently.

The ordinary LVQ algorithm does not have a limit of distance through which a weight vector can approach an input vector. Therefore, the modified LVQ algorithm imposes a limit on how close a weight vector can approach an input vector. This limit is realized by changing the learning rate appropriately as the learning proceeds, and the limit depends on the amount of the reward that the action gets. Therefore, an incorrect weight vector can not be closer to the input vector than the correct weight vector (Fig.3).

The modified learning algorithm can be expressed by Eq.(3)

$$w_{(k+1)} = w_{(k)} + \alpha_{(k)} s_{(k)} r_{(k)} (x_{(k)} - w_{(k)}), \quad (3)$$

where,

k : learning iteration number,

$\alpha_{(k)}$: learning rate,

$r_{(k)}$: amount of reward,

$s_{(k)} : \begin{cases} +1 & : \text{if reward is received,} \\ -1 & : \text{if punishment is received.} \end{cases}$

Using Eq.(3), $w_{(k+2)}$ can be calculated as

$$w_{(k+2)}$$

$$\begin{aligned} &= (1 - \alpha_{(k+1)} s_{(k+1)} r_{(k+1)}) w_{(k+1)} \\ &\quad + \alpha_{(k+1)} s_{(k+1)} r_{(k+1)} x_{(k+1)} \\ &= (1 - \alpha_{(k+1)} s_{(k+1)} r_{(k+1)}) (1 - \alpha_{(k)} s_{(k)} r_{(k)}) w_{(k)} \\ &\quad + (1 - \alpha_{(k+1)} s_{(k+1)} r_{(k+1)}) \alpha_{(k)} s_{(k)} r_{(k)} x_{(k)} \\ &\quad + \alpha_{(k+1)} s_{(k+1)} r_{(k+1)} x_{(k+1)}. \end{aligned} \quad (4)$$

For a constant λ , by setting $\alpha_{(k)}$ and $\alpha_{(k+1)}$ so that

$$\begin{aligned} &|\alpha_{(k+1)} s_{(k+1)} r_{(k+1)}| \\ &= \lambda |1 - \alpha_{(k+1)} s_{(k+1)} r_{(k+1)}| \alpha_{(k)} s_{(k)} r_{(k)}, \end{aligned} \quad (5)$$

the contribution of $x_{(k+1)}$ in determining $w_{(k+2)}$ becomes as λ times strong as that of $x_{(k)}$. Because Eq.(3) is linear with respect to $x_{(k)}$, if Eq.(5) is satisfied for any k , the contribution of $x_{(k)}$ will be as λ^k times strong as that of $x_{(0)}$.

If $0 < \lambda < 1$, the contribution of the input vector $x_{(k)}$ decreases as the learning proceeds (as $k \rightarrow \infty$).

Since $|s| = 1, 0 < \alpha_{(k)}, 0 < r_{(k)}$, Eq.(5) is written as

$$\alpha_{(k+1)} r_{(k+1)} = \lambda |1 - \alpha_{(k+1)} s_{(k+1)} r_{(k+1)}| \alpha_{(k)} r_{(k)}.$$

Especially if the learning rate is chosen so that $0 < \alpha_{(k)} r_{(k)} < 1$, we get from Eq.(5)

$$\alpha_{(k+1)} = \frac{\lambda \alpha_{(k)} r_{(k)}}{(1 + \lambda s_{(k+1)} r_{(k+1)} \alpha_{(k)}) r_{(k+1)}}, \quad (6)$$

and therefore

$$\alpha_{(k)} = \frac{\lambda^k \alpha_{(0)} r_{(0)}}{(1 + r_{(0)} \alpha_{(0)} \sum_{j=1}^k \lambda^j s_{(j)}) r_{(k)}}. \quad (7)$$

If $r_{(0)} = 1$ and $\lambda = 1$, Eq.(3) and (6) are reduced to Kohonen's OLVQ1 algorithm. Now, let us consider a situation where the algorithm accepts the same input vector x repeatedly.

By letting $x_{(k)} = x, r_{(k)} = r$ and $s_{(k)} = s$, we get

$$\begin{aligned} w_k &= [(1 - sr\alpha_{(k-1)})(1 - sr\alpha_{(k-2)}) \cdots (1 - sr\alpha_0)] w_0 \\ &\quad + [(1 - sr\alpha_{(k-1)}) \cdots (1 - sr\alpha_1) sr\alpha_0] \\ &\quad + (1 - sr\alpha_{(k-1)}) \cdots (1 - sr\alpha_2) sr\alpha_1 \\ &\quad + \cdots \\ &\quad + (1 - sr\alpha_{(k-1)}) sr\alpha_{(k-2)} \\ &\quad + sr\alpha_{(k-1)}] x. \end{aligned} \quad (8)$$

Combining Eq.(6),(7) and (8), we get,

$$w_{(k)} = \frac{1 - sr\alpha_{(0)}}{1 + sr\alpha_{(0)} \sum_{j=1}^{k-1} \lambda^j} w_{(0)} + \frac{sr\alpha_{(0)} \sum_{j=0}^{k-1} \lambda^j}{1 + sr\alpha_{(0)} \sum_{j=1}^{k-1} \lambda^j} x. \quad (9)$$

Fig. 4 Distance between input vector and weight vector after training

Fig. 5 Relation between $r s \alpha_{(0)}$ and d

The detailed derivation of Eq.(9) is written in Appendix.

Thus, if $0 < \lambda < 1$,

$$\lim_{k \rightarrow \infty} \mathbf{w}_{(k)} = \frac{(1-\lambda)(1-rs\alpha_{(0)})}{1-\lambda+rs\alpha_{(0)}\lambda} \mathbf{w}_{(0)} + \frac{rs\alpha_{(0)}}{1-\lambda+rs\alpha_{(0)}\lambda} \mathbf{x}. \quad (10)$$

From Eq.(10), we can represent the distance between the final weight vector and the input vector in terms of the distance between the initial weight vector and the input vector

$$\mathbf{x} - \lim_{k \rightarrow \infty} \mathbf{w}_k = d(\mathbf{x} - \mathbf{w}_{(0)}), \quad (11)$$

where

$$d = \frac{1-\lambda}{\frac{1}{1-rs\alpha_{(0)}} - \lambda}. \quad (12)$$

In Fig.4 the weight vector cannot approach the input vector beyond the point where the distance between them is d .

Therefore, even if the same input vector is repeatedly input to the algorithm, the weight vector cannot approach the input vector indefinitely. There is a limit on how close the weight vector can approach the input vector thanks to the learning rate Eq.(6).

The limit d depends on the amount of reward assigned to the input vector $\mathbf{x}$. Relation between $r s \alpha_{(0)}$ and d is shown in Fig.5. When $s = 1$ and reward is big, d will be small. As the received reward becomes larger, the weight vector can be closer to the input vector.

4. Path finding in maze

The following path finding problem is treated where a

Fig. 6 Action of Robot

Fig. 7 Input of LVQ network from maze

Select of optimal action

Fig. 8 Structure of LVQ network

robot searches for the shortest path from the start point to the goal in a maze by the modified LVQ algorithm.

The robot senses its present position(x, y) and the existence of obstacles in its neighborhood, and chooses its action among 'go-forward', 'go-right', 'go-left' and 'go-backward'(Fig.6). Therefore the input vector contains position of robot(x_1, x_2), and existence/absence of obstacles in north(x_3), east(x_4), south(x_5) and west(x_6) cells(See Fig.7).

The modified LVQ network (Fig.8) has been applied to a path finding problem in a 20 x 20 maze(Fig.9). The optimal path from the start point to the goal in the maze is 38 steps in length. Here, in one step, the robot moves from the present cell to the next cell.

Starting from the start point, the robot goes through obstacles and tries to reach the goal. During exploring

Fig. 9 Maze used in the test

Fig. 10 ϵ as a function of episodes

the maze, the robot selects its action based on ϵ -greedy strategy¹⁾. It does not always select the greedy action but selects other action at a probability of ϵ , and performs the selected action. The robot can explore more widely by this method in the maze. Values of ϵ used in this example is shown by Fig.10.

We update the weight vectors after every episode. An episode is a period during which the robot sets off the start point and ends in a terminal state. There are two terminal states. One is the state where the robot reaches the goal, and the other is the time-out state. When the robot has successfully arrived at the goal, the weight vectors of the LVQ output nodes activated along the path to the goal are updated by Eq.(13),(14)

$$w_{(k+1)} = w_{(k)} + \alpha_{(k)} r_{(k)} s (x_{(k)} - w_{(k)}), \quad (13)$$

$$s = 1, \quad (14)$$

$$r_{(k)} = \frac{1}{total_step_{(k)} - \beta}.$$

where k is the episode number, $\alpha_{(k)}$ is calculated by Eq.(6) with $\alpha_{(0)}=0.05$ and $total_step_{(k)}$ is the number of moves that the robot has made in this episode k . The reward $r_{(k)}$ is equally given to all the actions in this episode. If $total_step_{(k)}$ is small, i.e. the robot finds the goal

Table 1 The number of steps of the best path

Random number sequence	LVQ algorithm	Proposed LVQ algorithm
1	Failed	38 steps
2	42 steps	38 steps
3	54 steps	38 steps
4	40 steps	38 steps
5	52 steps	38 steps
6	42 steps	38 steps
7	44 steps	42 steps
8	Failed	52 steps
9	38 steps	44 steps
10	42 steps	42 steps

quickly, the robot gets a large reward $r_{(k)}$. This, combined with the modified LVQ, drives the robot to find the shortest path. Coefficient β adjusts the relation between $total_step_{(k)}$ and $r_{(k)}$. Here β is 30 considering the length of the shortest path is 38. When the robot cannot reach the goal within a prespecified number of steps N , the reward $r_{(k)}$ and its sign s are set as

$$\begin{aligned} r_{(k)} &= 0.05, \\ s &= -1. \end{aligned} \quad (15)$$

The path finding algorithm for one episode can be summarized as follows:

- 1: Receive the input vector $[x_1, x_2, \dots, x_6]^T$.
- 2: Search for the weight vector which minimizes d_j , and find the best action at this time.
- 3: Select the actual action by ϵ -greedy strategy.
- 4: Move.
- 5: Locate the present position.
If the position is goal, then update the weight vectors by Eq.(13),(14).
If the position is not goal and $total_step_{(k)} > N$, then update the weight vectors by Eq.(13),(15).
- 6: $total_step_{(k)} := total_step_{(k)} + 1$
Go to 1.

Training was done for 800 episodes with 10 different randomly initialized weights.

Table 1 shows the length of the shortest path that was found by the Kohonen's LVQ algorithm and the modified LVQ algorithm. In Table 1, the ordinary LVQ algorithm can hardly find the optimal path to the goal, while the modified LVQ algorithm can in almost all cases successfully find the optimal path of 38 steps in length to the goal. More specifically, the ordinary LVQ algorithm found a path to the goal in 8 out of 10 cases, but found the optimal path just in a single case. But the modified LVQ algorithm successfully found a path to the goal in all the cases, and found the optimal path in 6 cases.

Table 2 The number of total steps until find optimal path is found

Random number sequence	Proposed LVQ algorithm	Q-learning algorithm
1	58123 steps	159209 steps
2	39481 steps	
3	32209 steps	
4	32802 steps	
5	42708 steps	
6	34009 steps	
7	39423 steps	
8	51845 steps	
9	76683 steps	
10	40120 steps	

Table 2 compares the results by Q-learning and modified LVQ algorithm. Table 2 shows the total number of steps until the optimal path is found. Results by Q-learning shown in the table is the best among those obtained with different Q-learning parameter settings. The table indicates that the proposed algorithm is much faster than Q-learning algorithm.

5. Conclusions

A method has been presented for searching for the optimal paths in mazes which have many paths from start point to goal. The method is based on newly devised modified Learning Vector Quantization (LVQ) in reinforcement learning framework. The modified LVQ algorithm limits the distance that a weight vector can be close to input vector to cope with erroneous teacher signals. From the results of the example problem, it has been found that the modified LVQ performs much better than the ordinary LVQ: the ordinary LVQ can find paths from the start point to the goal, but it hardly finds the optimal path, on the other hand, the modified LVQ finds the optimal path at a high probability. Learning time of Q-learning is long because the Q-learning calculates Q-values of all available actions and chooses the best action. But LVQ algorithm chooses directly the best action. From the result of example problem, the LVQ algorithm can learn the shortest path quickly compared with Q-learning.

This certifies the validity and usefulness of the proposed LVQ-based reinforcement learning method.

APPENDIX

Let us recite Eq.(8)

$$\begin{aligned}
 w_k &= [(1 - sr\alpha_{(k-1)})(1 - sr\alpha_{(k-2)}) \cdots (1 - sr\alpha_0)]w_0 \\
 &\quad + [(1 - sr\alpha_{(k-1)}) \cdots (1 - sr\alpha_1)]sr\alpha_0 \\
 &\quad + (1 - sr\alpha_{(k-1)}) \cdots (1 - sr\alpha_2)]sr\alpha_1
 \end{aligned}$$

$$\begin{aligned}
 &+ \cdots \\
 &+ (1 - sr\alpha_{(k-1)})sr\alpha_{(k-2)} \\
 &+ sr\alpha_{(k-1)}]x.
 \end{aligned} \tag{16}$$

Here, $(1 - sr\alpha_{(k)})$ is calculated as follows by using $\alpha_{(k)}$ in Eq.(7),

$$\begin{aligned}
 1 - sr\alpha_{(k)} &= 1 - \frac{1 - sr\lambda^k\alpha_0}{1 + sr\alpha_0 \sum_{j=1}^k \lambda^j} \\
 &= \frac{1 + sr\alpha_0 \sum_{j=1}^{k-1} \lambda^j}{1 + sr\alpha_0 \sum_{j=1}^k \lambda^j}.
 \end{aligned} \tag{17}$$

Using Eq.(17), the coefficient of w_0 in Eq.(16) is calculated as follows

$$\begin{aligned}
 &(1 - sr\alpha_{(k-1)})(1 - sr\alpha_{(k-2)}) \cdots (1 - sr\alpha_1)(1 - sr\alpha_0) \\
 &= \frac{1 - sr\alpha_0}{1 + sr\alpha_0 \sum_{j=1}^{k-1} \lambda^j}.
 \end{aligned} \tag{18}$$

The coefficient of x in Eq.(16) is calculated as follows using Eq.(5),

$$\begin{aligned}
 &(1 - sr\alpha_{(k-1)}) \cdots (1 - sr\alpha_1)sr\alpha_0(1 + \lambda + \cdots + \lambda^{(k-1)}) \\
 &= \frac{sr\alpha_0 \sum_{j=0}^{k-1} \lambda^j}{1 + sr\alpha_0 \sum_{j=1}^{k-1} \lambda^j}.
 \end{aligned} \tag{19}$$

Therefore, $w_{(k)}$ is as following,

$$w_{(k)} = \frac{1 - sr\alpha_0}{1 + sr\alpha_0 \sum_{j=1}^{k-1} \lambda^j} w_{(0)} + \frac{sr\alpha_0 \sum_{j=0}^{k-1} \lambda^j}{1 + sr\alpha_0 \sum_{j=1}^{k-1} \lambda^j} x. \tag{20}$$

References

- 1) R.S. Sutton and A.G. Barto: Reinforcement Learning. An introduction. MIT press, Cambridge, MA. (1998)
- 2) C.J.C.H. Watkins and P. Dayan: Technical Note: Q-Learning, *Machine Learning*, **8**, 55/68 (1992)
- 3) T. Kohonen: Self-Organizing Maps, Springer Series in Information Sciences (1995)
- 4) T. Kohonen: Learning Vector Quantization for Pattern Recognition. Technical Report TTK-F-A602, Helsinki University of Technology, Finland (1990)
- 5) T. Kohonen: The learning vector quantization program package. Finland, Helsinki University of Technology (1993)
- 6) M. Hagiwara: Neuro, Fuzzy, Genetic Algorithms (in Japanese) (1994)

Shon MIN KYU

Received the B.S. degree in Electrical Engineering from Myong Ji University, Korea, in 1998, and M.S degree in Graduate School of Information Science and Electrical Engineering, Kyushu University, in 2000.

Now he is a second-year Dr. candidate at Graduate School of Information Science and Electrical Engineering, Kyushu University.

Junichi MURATA (Member)

Received the Master and Doctor of Engineering degrees from Kyushu University, Japan, in 1983 and 1986, respectively.

He then became a Research Associate and an Associate Professor at Graduate School of Information Science and Electrical Engineering, Kyushu University. His current research interests are neural networks, self-organizing systems and their applications to control and identification.

Dr. Murata is a member of ISCIE, IEEJ and IEEE

Kotaro HIRASAWA (Member)

Received the B.S. degree and the M.S degree in electrical engineering from Kyushu University, Japan in 1964 and 1966, respectively. From 1966 to 1992, he was with Hitachi Ltd., where he served as a Vice Present at the Hitachi Research Laboratory. Since December 1992, he has been a Professor in the Faculty of Engineering, Kyushu University. he is currently with the Electrical Engineering at the same university.

Dr. Hirasawa is a member of IEEE.