Int. J. Fuzzy Syst. (2019) 21(1):95–103
https://doi.org/10.1007/s40815-018-0535-y

An Ensemble Fuzzy Approach for Inverse Reinforcement Learning

Wei Pan[1] · Ruopeng Qu[1] · Kao-Shing Hwang[2] · Hung-Shyuan Lin[2]

Received: 11 April 2018 / Revised: 24 July 2018 / Accepted: 31 July 2018 / Published online: 20 August 2018

Abstract In reinforcement learning, a reward function is a priori specified mapping that informs the learning agent how well its current actions and states are performing. From the viewpoint of training, reinforcement learning requires no labeled data and has none of the errors that are induced in supervised learning because responsibility is transferred from the loss function to the reward function. Methods that infer an approximated reward function using observations of demonstrations are termed inverse reinforcement learning or apprenticeship learning. A reward function is generated that reproduces observed behaviors. In previous studies, the reward function is implemented by estimating the maximum likelihood, Bayesian or information theoretic methods. This study proposes an inverse reinforcement learning method that has an approximated reward function as a linear combination of feature expectations, each of which plays a role in a base weak classifier. This approximated reward function is used by the agent to learn a policy, and the resultant behaviors are compared with an expert demonstration. The difference between the behaviors of the agent and those of the expert is measured using defined metrics, and the parameters for the approximated reward function are adjusted using an ensemble fuzzy method that has a boosting classification. After some interleaving iterations, the agent performs similarly to the expert demonstration. A fuzzy method is used to assign credits for the rewards in respect of the most recent decision to the neighboring states. Using the proposed method, the agent approximates the expert behaviors in fewer steps. The results of simulation demonstrate that the proposed method performs well in terms of sampling efficiency.

✉ Kao-Shing Hwang
hwang@ccu.edu.tw

[1] School of Computer Science, Northwestern Polytechnical University, Xi'an, China

[2] Department of Electrical Engineering, National Sun Yat-Sen University, Kaohsiung, Taiwan

Keywords Inverse reinforcement learning · Fuzzy · Reward feature

1 Introduction

The objective of inverse reinforcement learning (IRL) is to learn a task using an approximation of an unknown reward by observing an expert demonstration [1, 2]. Many IRL methods offer frameworks for estimating a reward function, which is then used to derive an optimal policy [3, 4]. A Bayesian model provides a higher reward to the states that are close to the goal so there is greater adaptability to changes of task, such as the assignment of a new target or variations in parameters [5, 6]. A fuzzy comprehensive evaluation method is proposed to extract the features utilized in fuzzy Bayesian reinforcement learning and classifies the situations into a set of features. The factors defined for them can be used to calculate the weights [7]. These methods learn a mapping by observing an input to state values/costs using margin maximization techniques and minimize the cost function using gradient descent or game-theoretic approach [8]. The residual gradient fuzzy reinforcement learning algorithm uses the residual gradient algorithm with fuzzy actor–critic structure to tune the input and output parameters of its function approximation systems [9]. The maximum entropy algorithm [10, 11] uses a Markov decision process (MDP) model to calculate a probability distribution for the state actions. These IRL methods assume that a MDP model of a system is either given a priori or can be accurately learned using the demonstrated trajectories, in terms of a state space. Nevertheless, the IRL concept by matching of a defined feature expectation is ambiguous. Each policy can be optimal for many reward functions, and many policies lead to the same feature counts.

A notable exception is the projection algorithm, whereby the trajectories that are generated by the learning agent are compared with the demonstrated trajectories using state features, in order to minimize the worst-case loss in the value of the learned policy. This guarantees that the correct reward function is learned if the feature counts are properly estimated. The algorithms regard an IRL problem as an approximation of a linear combination function with constraints. Using the orthogonal projection method to calculate the weights for the linear reward function, the method estimates the linear combination function and derives an optimal policy faster and with less effort [2]. From the viewpoint of supervised learning, this type of IRL uses supervised learning techniques to acquire the reward function. Supervised learning uses labeled data to approximate a mapping between inputs and outputs. If labeled data (features) are used to derive a reward function, an agent must explore the features space and derive an optimal action policy. The human reinforcement learning links direct policy evaluation and human reinforcement to autonomous robots, accordingly shaping the reward function by combining both reinforcement signals [12]. Besides, the architecture of adaptive heuristic critic combined with action bias allows agents to solve the problem of continuous action system [13]. Unfortunately, it is likely for this simple method to become trapped by problems or failures [6].

Using ensemble learning techniques, the proposed approach inputs the compared trajectories for IRL into a boosting classification problem and provides a faster and more flexible method of adjusting the weights of the approximated reward function comparing with the aforementioned methods [14, 15]. Besides, in order to improve the performance of RL algorithm, fuzzy-based methods were proposed for tuning the learning rate [16, 17]. In this paper, the proposed approach also uses a fuzzification technique to accelerate the updating process. The remainder of this paper is organized as follows. Section 2 briefly introduces the relative research background, including reinforcement learning, inverse reinforcement learning and the boosting classifier. The proposed ensemble inverse reinforcement learning technique that uses boosting and fuzzification techniques is introduced in Sect. 3. In Sect. 4, the simulation results for a labyrinth and a robotic soccer are used to demonstrate the validity of the proposed method. The final section draws conclusions.

2 Background

2.1 Inverse Reinforcement Learning

In some IRL algorithms, the reward function is represented by a linear combination of useful features as [14]:

$$R(s) = \omega \cdot \phi(s) \tag{1}$$

where $\phi(s) = [\phi_1(s), \phi_2(s), \ldots, \phi_k(s)]^T$ is a function mapping from a state space to a feature vector that is predefined as 0 or 1, k is the number of features in the reward function, and $\omega(s) = [\omega_1, \omega_2, \ldots, \omega_k]$ is an unknown vector to be tuned during the learning process. The value function of policy π is shown below. The expected discounted accumulated feature value is defined as a vector of feature expectations in Eq. (3):

$$\begin{aligned} E_{s_0 \sim D}[V^{\pi}(s_0)] &= E\left[\sum\nolimits_{t=0}^{\infty} \gamma^t R(s_t) | \pi\right] \\ &= E\left[\sum\nolimits_{t=0}^{\infty} \gamma^t \omega \cdot \phi(s_t) | \pi\right] \\ &= \omega \cdot E\left[\sum\nolimits_{t=0}^{\infty} \gamma^t \phi(s_t) | \pi\right] \end{aligned} \tag{2}$$

$$\mu(\pi) = E\left[\sum\nolimits_{t=0}^{\infty} \gamma^t \phi(s_t) | \pi\right] \tag{3}$$

IRL estimates a reward function that allows the agent to generate the same trajectory as the demonstrated trajectory using a policy under this approximate reward function.

2.2 Boosting Classifier

Ensemble methods are learning algorithms that construct a set of classifiers and then classify new data points by taking a weighted vote of their predictions. Some common ensemble methods use Bayesian averaging and error-correcting output coding. Boosting algorithms, which correct the error in output coding, and convert weak learners to proficient learners, improve the accuracy of prediction in a machine learning model when they are combined with a learning algorithm. A boosting algorithm constructs a strong classifier that is a linear combination of weak classifiers with adaptive weights for training data, as defined below, where α_t denotes the weight of the selected weak classifier at iteration t, and $h_t(x)$ denotes the output of the selected weak classifier at iteration t using training data x.

$$F_T(x) = \sum_{t=1}^{T} \alpha_t h_t(x) \tag{4}$$

At each iteration, t, the boosting algorithm selects a weak classifier and adjusts weight α_t for this weak classifier to minimize the sum of training error, E_t, for the boosted classifier that is defined in Eq. (5). $|X|$ denotes the number of samples in the training dataset, $F_{t-1}(x)$ is the boosted classifier that constructed by iteration *t-1,* and *E(F)* denotes some error function.

$$E_t = \sum_{i=1}^{|X|} E(F_{t-1}(x_i) + \alpha_t h_t(x_i)) \quad (5)$$

2.3 The AdaBoost Algorithm and the Concept of Dissimilarity

AdaBoost is an abbreviation for adaptive (adjust the error rate) and boost (improve the classification accuracy). It is implemented in the framework of boosting algorithm [18, 19]. AdaBoost calculates the adjustment parameter α using the error rate of each classifier and combines the weak classifiers into a strong classifier [20]. Figure 1 shows a simple schematic diagram of AdaBoost. The classifiers decide to which category the data belong, and the parameter, α, defines the classifiers' credibility. By combining the classifiers and the parameter, α, data are classified sequentially.

The original AdaBoost algorithm and the AdaBoost.M1 algorithm are slightly different. AdaBoost.M1 updates the weights via a training process. The input data have m points, $x_1, x_2, \ldots, x_m$. Each point corresponds to classification labels $y_1, y_2, \ldots, y_m$, $y_k \in \{1,2,\ldots,g\}$. After i iterations, the value for the weight of sample $x_k \in \{x_1, x_2, \ldots, x_m\}$ is expressed as $D_i(k)$. The initialized weight values are all the same at $1/m$. After training, the values of the weights for correct classifications are reduced and the weights for incorrect classifications are increased. If h_i represents the classifier that is selected at the *i*th iteration and $h_i(x_k) \neq y_k$ denotes that the output is not the same as actual y_k, the classification is incorrect. The samples are individually checked in the classifier to determine the correctness of each classification. The error rate E_i is added all of the weight values of the incorrectly classified samples, as shown in formula (6). The parameter, α_i, is defined in formula (7).

$$E_i = \sum_{k:h_i(x_k)\neq y_k} D_i(k) \quad (6)$$

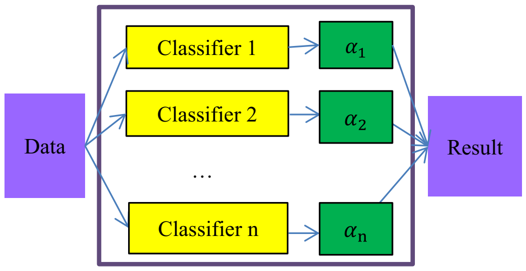


Fig. 1 A schematic diagram of an AdaBoost

$$\alpha_i = \ln\left(\frac{1-E_i}{E_i}\right) \quad (7)$$

The larger the error rate E_i, the smaller is the parameter α_i and the less reliable is the classifier.

The iterative and regularized formulation of $D_i(k)$ is depicted as:

$$Z_i = \sum_m D_{i+1}(k)$$
$$D_{i+1}(k) = \frac{D_i(k)}{Z_i} * \left\{ \begin{array}{ll} 1 & \text{if } y_k \neq h_i(x_k) \\ e^{-\alpha_i} & \text{if } y_k = h_i(x_k) \end{array} \right\} \quad (8)$$

3 Proposed Method

IRL establishes a reward function, *R*, which can generate a policy that is similar to the expert's policy. The difference between the expert's behaviors and the agent's behaviors is used to update the reward function over many iterations. When the reward function has been adjusted, the agent learns a policy that is similar to the expert's behavior.

3.1 Ensemble Techniques for Inverse Reinforcement Learning

If an ensemble method is used for inverse reinforcement learning, the feature expectations defined in (9) are regarded as a set of week classifiers and the trajectories acquired from demonstrations are used as class labels for the training sample, i.e., the desired outputs. To calculate the coefficients for the weak classifiers, the proposed approach uses the framework of an ensemble method for IRL, to determine the importance of the distribution of features that causes policy value errors in a search for iterative policy [21, 22]. Therefore, the parameters for a linear combination of feature functions are updated using the importance distribution as the step size and the errors as the direction for each iteration. In other words, the proposed method uses the difference between the expert behaviors and the agent behaviors to determine the degree of importance for features and adjusts the weights for the weak classifier. During successive learning iterations, the parameters for the linear combination of the reward function are updated as:

$$\omega_{i+1} = \omega_i + D_{i+1} \times \Delta\mu_i \quad (9)$$

where ω_i is the value of the weight, *i* is the number of iterations, and *D* is the ensemble voting weight, as previously defined before. The difference between the expert trajectory and that which is generated using the learned policy that is based on the transiently approximated reward function is $\Delta\mu_t = \mu(\pi_E) - \mu(\pi_t)$. $\mu(\pi_E)$ is the target feature expectation vector for the corresponding sample inputs from the demonstration, and $\mu(\pi t)$ is the contemporary feature expectation vector, which is acquired from the policy that uses a transiently approximated reward function at iteration t.

Equations (1) and (9) define the optimum weight value ω^* as a set of linear combinations. The proposed ensemble method for determining the values of parameters using this method is listed in Algorithm 1 [2]. The algorithm shows the step for updating the value of ω and for finding the value of μ in IRL methods. E_i represents the error rate, which is calculated by adding all of the categorization errors. The error rate is used to adjust each element of the value of the weight. If the error rate is high, the value of the weight is decreased. If the error rate is small, the value of the weight is increased.

```
**Initialize** $\omega_i(s) = 0 \quad for\ all\ s \in S,\ D_1(k) = \frac{1}{m}\ and\ i = 0$ ,
**Repeat for each iteration**
    1. $i = i + 1$
    2. According to RL algorithm, compute the optimal policy $\pi_i$ using the rewards R defined in equation (1)
    3. Compute the expected discounted accumulated feature value $\mu_i$ according to equation(3)
    4. $E_i = \sum_{k:\mu_i^k \neq \mu_E^k} D_{i+1}(k).$
    5. $D_{i+1}(k) = \frac{D_i(k)}{Z_i} \times \begin{cases} D_i(k)e^{-\alpha_i} & ,if\ \mu_e(k) = \mu_i(k) \\ 1 & ,if\ \mu_e(k) \neq \mu_i(k) \end{cases}$
    6. $Normalize\ D_{i+1}(k).$
    7. $\omega_{i+1} = \omega_i + D_{i+1} \times \Delta\mu_i$
```

Algorithm 1 The algorithm for the ensemble method

3.2 Inverse Reinforcement Learning Using Dissimilarity

Since the ensemble method requires many learning iterations, this paper proposes an ensemble method that uses dissimilarity to adjust the value of ω in the vicinity of a specific future using batch learning [23]. In the first step, the weight for the reward is generated at a random value, ω_0, to calculate the expected characteristic value μ_i using the policy π_i which is generated by Eq. (1) in the IRL. The characteristic expected value μ_E is the expert behavior.

Comparing μ_i with μ_E, the following three conditions are possible:

1. An element in μ_i is larger than that in the corresponding position in μ_E, which means that the route that is learned by the agent passes earlier than the expert route.
2. An element in μ_i is smaller than that in the corresponding position in μ_E, which means that the route that is learned by agent passes later than the expert route.
3. All elements in μ_i are as the same as those in the corresponding position in μ_E, which means that the route that is learned by the agent passes at the same time as the expert route.

The difference between the agent's behaviors and expert's behaviors is calculated as:

$$\Delta\mu_i = \mu_E - \mu_i \tag{10}$$

This paper uses the concept of dissimilarity to update the learning rate for the elements in ω_i. Because the goal is to emulate the expert exactly, adjustments are made using the dissimilarity between the expert's and the agent's paths.

The relevant equation is:

$$S_j = \sqrt{\frac{\left(\frac{D_{1,e_1}-D_{1,a_1}}{D_{1,range}}\right)^2 + \cdots + \left(\frac{D_{j,e_j}-D_{j,a_j}}{D_{j,range}}\right)^2 + \cdots + \left(\frac{D_{n,e_n}-D_{n,a_n}}{D_{n,range}}\right)^2}{n}} \tag{11}$$

where S_j means the reward features, $j = 1, 2,\ldots, n$. $D_{j,e}$ is the experts' weight in dimension i, $D_{j,a}$ represents the learning agent's weight in dimension i, and $D_{j,range}$ represents the maximum range of the weights in dimension i.

Equation (11) allows the agent's behaviors to emulate the expert's using fewer learning iterations. The weight for the reward is:

$$\omega_{i+1} = \omega_i + S_{i+1} \times \Delta\mu_i \tag{12}$$

If the dissimilarity S_i is high, the weight for the reward must be significantly adjusted, and if it is low, little adjustment is required.

3.3 IRL with Fuzzification

In IRL, the policy updates the reward feature ω_i using Eq. (12). The value for $\Delta\mu_i$ represents the difference between the expert's and the agent's behaviors. The learning process is illustrated in Fig. 2.

IRL adjusts the weight for the reward ω_i, which is based on μ_i. This is then used to establish a new policy. The learning agent uses the policy to learn behaviors. To accelerate learning further, fuzzification and dissimilarity are used to learn the expert's behaviors quickly. An example of a robotic soccer game, which is shown in Fig. 3, is used to illustrate the implementation of dissimilarity and fuzzification of the state space. In a robotic soccer game, the state space is composed of the two-dimensional one-spanned angle between the robot and the ball and the distance between the robot and ball. The dimension of the angle is divided into four portions, {(0°, 90°), (90°, 180°), (180°, 270°), (270°, 360°)}, and the distance is partitioned into four ranges, {(0, 1), (1, 2), (2, 3), (2, 3)}, so the cardinality of state space is 16; that is, there are 16 discrete states. Triangular membership functions are used for fuzzification, as shown in Fig. 4. If the current sensory input is (1.3°, 99°), the weight of the current state containing (1.3°, 99°) is updated with the sum of the weights for the active vicinity: the top left four grids on the table in Fig. 5; i.e., $0.08 \times \omega_1 + 0.12 \times \omega_2 + 0.32 \times \omega_5 + 0.48 \times \omega_6$.

There are two possible outcomes when the value for $\Delta\mu_i$ is calculated over the state space. If $\Delta\mu_i$ is negative, i.e., the expert's expected feature μ_E is smaller than the agent's value for μ_i on the state i along the trajectory that is generated using the currently learned policy, the learning agent visits this state earlier using its own trajectory than if it is uses the expert's trajectory and vice versa. If $\Delta\mu_i = 0$, the learning agent and the expert either visit that state during the same time sequence or neither visits that state. The algorithm is more complex so the latter is the most common for an IRL process. There is no reward value of ω_i, and time is wasted in RL learning process. The fuzzification process allows an agent to update its own ω_i about rarely visited states by referring to a neighbor's experience, and learning is accelerated.

4 Simulations and Discussion

Three simulations were conducted to demonstrate the proposed method: a mountain car, a mobile robot in a maze and a robotic soccer game. The first three experiments compare the methods for this paper to verify the performance: the method used in [3] and the proposed IRL. The robotic soccer game simulation verifies that using fuzzy theory for the IRL algorithm improves the reinforcement learning rate.

4.1 Maze

In this simulation, a robot tries to determine a reward function, which is used to learn the demonstrator's strategy and approach the goal in a maze, as shown in Fig. 6. The simulation of maze problem shows the performance of the proposed IRL in discrete environments.

The maze has 20 × 20 grids, which are surrounded by walls. The yellow part in the maze is starting point for the

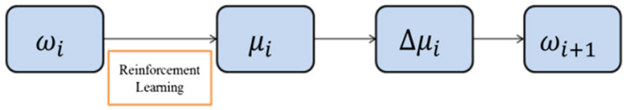


Fig. 2 The learning process of IRL

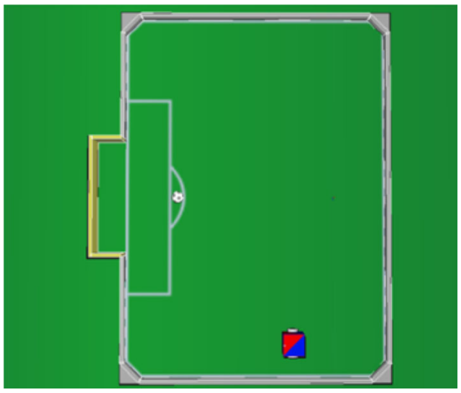


Fig. 3 An example of a soccer robot

mobile robot. The goal is the red rectangle in the maze. The gray parts represent the obstacles. The robot can move up, down, left or right. The robot can move only one grid at one time. The learning rate α was set to 0.7, and the ε-greedy action selection implemented in the simulation, that the exploration rate, was set to 5%. The other setting of parameters is listed in Table 1. After learning, the agent can imitate the expert's demonstration shown in Fig. 6.

The average iteration round time is listed in Table 2 to show the proposed IRL efficiency. It takes less time than the method in [3]. This demonstrates that the proposed method learns the demonstrated path more quickly. The proposed IRL's variance is smaller than its counterpart's, so that the proposed IRL is demonstrably more stable.

4.2 Mountain Car

This simulation, a standard testing domain in reinforcement learning, an under-powered car must drive up a steep hill, as shown in Fig. 7. Since gravity is stronger than the car's engine, even at full throttle, it must learn to leverage potential energy by driving up the opposite hill before the car is able to make it to the goal at the top of the rightmost hill. The dynamic equation is:

$$\begin{aligned} x_{t+1} &= x_t + \dot{x}_{t+1} \\ \dot{x}_{t+1} &= \dot{x}_t + 0.001a_t - 0.0025\cos(3x_t) \end{aligned} \tag{13}$$

where x_t is the position of the car and $\dot{x}_{t+1}$ is its velocity. The state space for the problem has two dimensions: its position and its velocity. The position interval is [− 1.2, 0.5], and the velocity interval is [− 0.07, 0.07]. In each state, three actions are allowed: throttle forward (+1), throttle reverse (−1) or zero throttles (0). The end of the simulation is when position ≥ 0.6.

The results in Table 3 show that after 100 rounds, the proposed IRL takes less time than its counterpart to complete the mountain car simulation. This demonstrates that the proposed method learns the path that expert demonstrates more quickly. The proposed IRL's variance is smaller than its counterpart's, which demonstrates that the proposed IRL is more stable.

4.3 Robotic Soccer Game Simulation

The simulation environment is the same as that shown in Fig. 3. Two-wheeled robot soccer must kick the ball into the goal in the same way as the expert. As shown in Fig. 8, the state space includes two dimensions: the angle of 18 ranges between the robot and ball, and the distance of 10 ranges between the robot and ball. The robot can move forward and backward and can rotate clockwise and anti-clockwise.

The results in Table 4 show that after 100 rounds, the proposed IRL requires fewer learning iterations than its counterpart. Table 4 shows that the proposed IRL also exhibits a smaller variance for the learning process than its counterpart, so it is more stable than its counterpart. The performance of the proposed IRL for behavior learning is validated in continuous environment, even though it is more difficult to design the reward function than in a discrete environment.

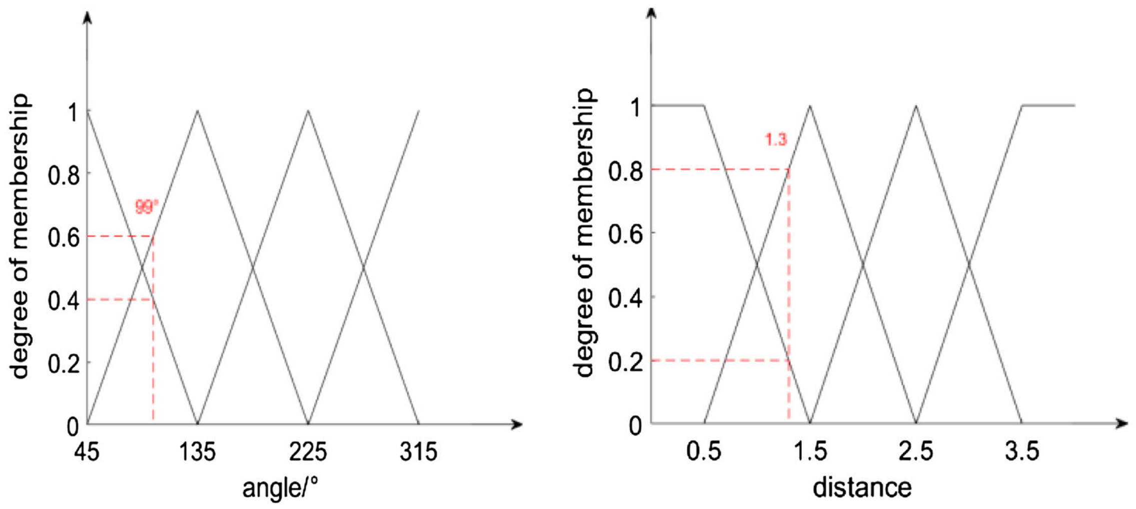
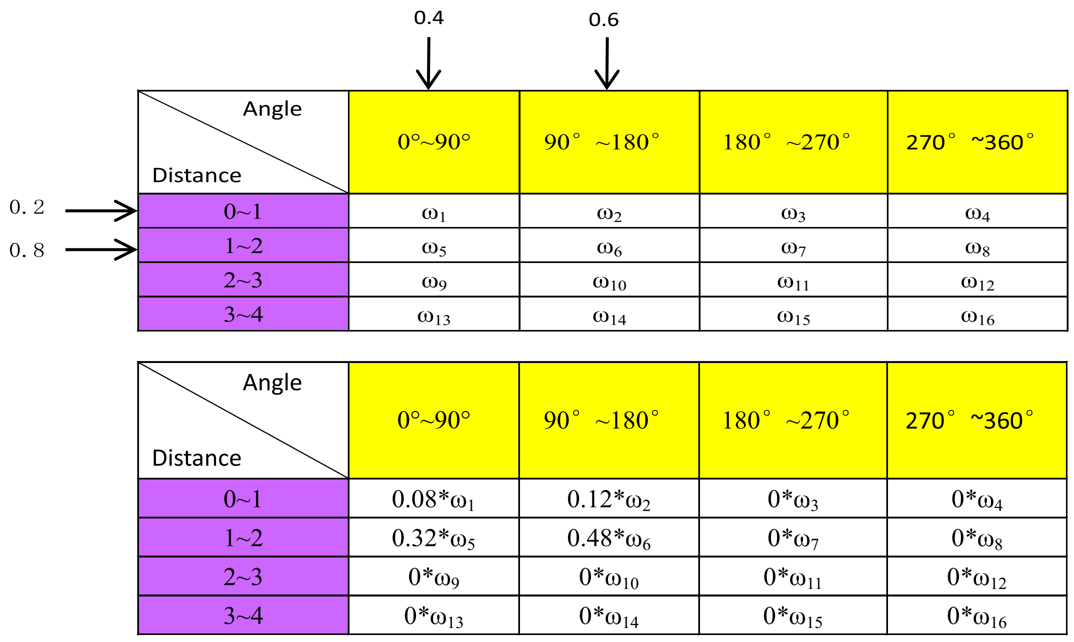


Fig. 4 The fuzzification example for robotic soccer games where the receptive field receives sensory data (1.3°, 99°)



Distance \ Angle	0°~90°	90° ~180°	180° ~270°	270° ~360°
0~1	ω_1	ω_2	ω_3	ω_4
1~2	ω_5	ω_6	ω_7	ω_8
2~3	ω_9	ω_{10}	ω_{11}	ω_{12}
3~4	ω_{13}	ω_{14}	ω_{15}	ω_{16}

Distance \ Angle	0°~90°	90° ~180°	180° ~270°	270° ~360°
0~1	$0.08*\omega_1$	$0.12*\omega_2$	$0*\omega_3$	$0*\omega_4$
1~2	$0.32*\omega_5$	$0.48*\omega_6$	$0*\omega_7$	$0*\omega_8$
2~3	$0*\omega_9$	$0*\omega_{10}$	$0*\omega_{11}$	$0*\omega_{12}$
3~4	$0*\omega_{13}$	$0*\omega_{14}$	$0*\omega_{15}$	$0*\omega_{16}$

Fig. 5 Weighted averages for the updated parameters in defuzzification

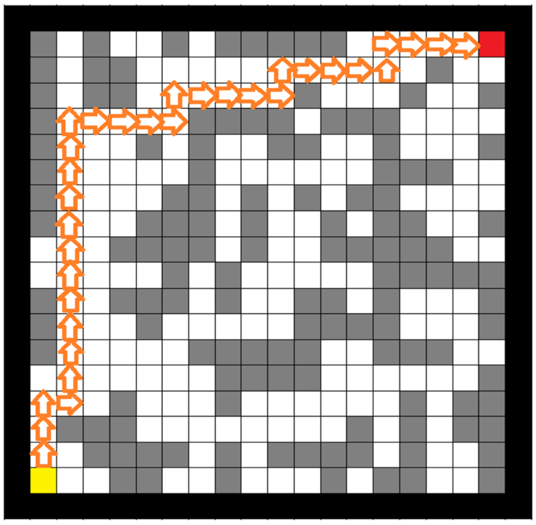


Fig. 6 The agent's path after learning

Table 1 Parameters of maze

Parameters	Values
α	0.7
γ	0.9
Exploration rate	5%
Strides	1 grid
Training round	200 round
Training step in one round	50 step

Table 2 Comparison of Number of Iteration between Two Methods in Maze

	Average iteration round time	SD
The counterpart	26.844	3.280
The proposed IRL	23.299	2.582

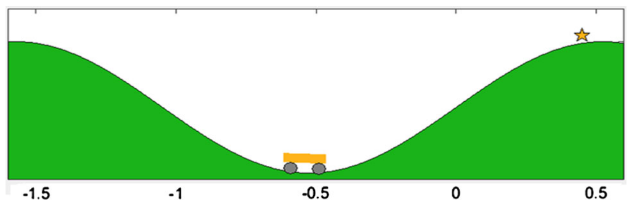


Fig. 7 Mountain car

4.4 Robotic Soccer Simulation with Fuzzy Theory

The simulation environment is the same as shown in Fig. 2. This simulation experiment verifies whether the proposed IRL-fuzzy can learn more quickly the same path as an expert. Table 5 shows the average number of iterations for the proposed IRL with and without fuzzy theory after 100, 200, 300, 400 and 500 rounds. It is seen the proposed IRL with fuzzy theory finds the reward R more quickly and saves much time.

Table 3 Comparison of number of iteration between two methods in mountain car

	Average iteration time	SD
The counterpart	74.421	8.203
The proposed IRL	63.672	5.956

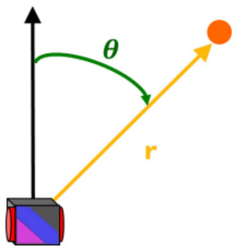


Fig. 8 Definition of state space

Table 4 Comparison of number of iteration between two methods in robot soccer

	Average iteration time	SD
The counterpart	85.974	9.956
The proposed IRL	73.722	6.620

5 Conclusions

This paper has proposed inverse reinforcement learning that uses the ensemble and fuzzy logic method. Using an approximated reward function, the agent learns the policy akin to the expert's demonstration. The learned policy that is derived from the approximated reward function can also accommodate stronger derivations, even though the agent may temporarily stray from the expert's demonstrated trajectory during exploration. The evolving policy leads the agent back to the trajectory. The proposed method relies on the concept of boosting methods for classification. In addition, a more advanced version of this method, using fuzzification, is introduced. The proposed fuzzy-based IRL method adjusts the weight for the reward to decrease the number of learning iterations. The method is implemented in robotic soccer simulation to verify the performance.

Table 5 Comparison of number of iteration between the proposed IRL with and without fuzzy theory

	100 rounds Iteration time	200 rounds Iteration time	300 rounds Iteration time	400 rounds Iteration time	500 rounds Iteration time
The proposed IRL	278.6	195.69	156.29	109.91	74.77
The proposed IRL-fuzzy	213.66	153.11	128.88	88.19	69.98

Even if a state space is complex, the feature weights are still updated to ensure more precise correction. Therefore, this method is particularly well suited for learning scenarios in the context of robotics.

Acknowledgements Research in this work was supported by the Seed Foundation of Innovation and Creation for Graduate Students in North-western Polytechnical University (No. zz2018166).

References

1. Zhifei, S., Joo, E.M." A review of inverse reinforcement learning theory and recent advances. In: 2012 IEEE Congress on Evolutionary Computation, Brisbane, QLD, pp. 1–8 (2012)
2. Hwang, K.S., Chiang, H.Y., Jiang, W.C.: Adaboost-like method for inverse reinforcement learning. In: 2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), Vancouver, BC, pp. 1922–1925 (2016)
3. Abbeel, P., Ng, A.Y.: Apprenticeship learning via inverse reinforcement learning. In: Proceedings of the 21st International Conference on Machine Learning, pp. 1–8 (2004)
4. Natarajan, S., Kunapuli, G., Judah, K., Tadepalli, P., Kersting, K., Shavlik, J.: Multi-agent inverse reinforcement learning. In: 2010 Ninth International Conference on Machine Learning and Applications, Washington, DC, pp. 395–400 (2010)
5. Sutton, R.S., Barto, A.G.: Reinforcement learning: an introduction. IEEE Trans. Neural Netw. **9**(5), 1054 (1998)
6. Ollis, M., Huang, W.H., Happold, M.: A Bayesian approach to imitation learning for robot navigation. In: 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, San Diego, CA, pp. 709–714 (2007)
7. Shi*, H., Lin, Z., Zhang, S., Li, X., Hwang, K.-S.: An adaptive decision-making method with fuzzy Bayesian reinforcement learning for robot soccer. Inf. Sci. **436–437**, 268–281 (2018)
8. Michini, B., Walsh, T.J., Agha-Mohammadi, A.A., How, J.P.: Bayesian nonparametric reward learning from demonstration. IEEE Trans. Robot. **31**(2), 369–386 (2015)
9. Awheda, M.D., Schwartz, H.M.: A residual gradient fuzzy reinforcement learning algorithm for differential games. Int. J Fuzzy Syst. **19**, 1058 (2017). https://doi.org/10.1007/s40815-016-0284-8
10. Syed, U., Schapire, R.: A game-theoretic approach to apprenticeship learning. In: Advances in Neural Information, Processing Systems, Vol. 20 (NIPS'08), pp. 1449–1456 (2008)
11. Ziebart, B., Bagnell, A., Dey, A.: Modeling interaction via the principle of maximum causal entropy. In: Proceedings of the Twenty-Seventh International Conference on Machine Learning (ICML'10), pp. 1255–1262 (2010)
12. Hwang, K.S., Lin, J.L., Shi, H., et al.: Policy learning with human reinforcement. Int. J. Fuzzy Syst. **18**, 618 (2016). https://doi.org/10.1007/s40815-016-0194-9
13. Hwang, K.S., Hsieh, C.W., Jiang, W.C., Lin, J.L.: A reinforcement learning method with implicit critics from a bystander. In: Advances in Neural Networks—ISNN 2017, pp. 363–270
14. Ng, A.Y., Russell, S.: Algorithms for inverse reinforcement learning. In: Proceedings of the 17th International Conference on Machine Learning, pp. 663–670 (2000)
15. Vapnik, V.N.: Statistical Learning Theory. Wiley, London (1998)
16. Shi, H., Li, X., Hwang, K.-S., Pan, W., Genjiu, X.: Decoupled visual servoing with fuzzy Q-learning. IEEE Trans. Ind. Inf. **14**(1), 241–252 (2018)
17. Pan, W., Lyu, M., Hwang, K-Sh, Ju, M.-Y., Shi, H.: A neuro-fuzzy visual servoing controller for an articulated manipulator. IEEE Access **6**(1), 3346–3357 (2018)
18. An, T.K., Kim, M.H.: A new diverse AdaBoost classifier. In: 2010 International Conference on Artificial Intelligence and Computational Intelligence, Sanya, pp. 359–363 (2010)
19. R.E. Schapire (2002) The boosting approach to machine learning an overview. In: MSRI Workshop on Nonlinear Estimation and Classification, Dec. 19, 2001, pp. 1–23 (2002)
20. Eibl, G., Pfeiffer, K.P.: How to make AdaBoost.m1 work for weak base classifiers by changing only one line of the code. In: Processing of the 13th European Conference on Machine Learning Helsinki, pp. 72–83 (2002)
21. Auer, P.: Using confidence bounds for exploitation-exploration trade-offs. J. Mach. Learn. Res. **3**, 397–422 (2002)
22. Browne, C.B., et al.: A survey of monte carlo tree search methods. IEEE Trans. Comput. Intell. AI Games **4**(1), 1–43 (2012)
23. Nicolescu, M., Jenkins, O.C., Olenderski, A.: Learning behavior fusion estimation from demonstration. In: ROMAN 2006—the 15th IEEE International Symposium on Robot and Human Interactive Communication, Hatfield, pp. 340–345 (2006)

Wei Pan is an Associate Professor in the School of Computer Science at Northwestern Polytechnical University, China, and visiting scholar of Electrical Engineering Department at National Sun Yat-sen University, Taiwan. He received his Ph.D. degree from Northwestern Polytechnical University, China, in 2008. His research interests include intelligent robots, machine learning, decision support systems and multi-agent systems.

Ruopeng Qu received the B.S. degree in computer science and technology from the School of Computer Science, Northwestern Polytechnical University, China, in 2012. He is currently pursuing the master's degree with the School of Computer Science in the same field. His research interests include intelligent robots, machine learning and decision support system.

Kao-Shing Hwang (M'93–SM'09) is a professor of Electrical Engineering Department at National Sun Yat-sen University and an adjunct professor of the Department of Healthcare Administration and Medical Informatic, Kaohsiung Medical University, Taiwan. He received the M.M.E. and Ph.D. degrees in Electrical and Computer Engineering from Northwestern University, Evanston, IL, USA, in 1989 and 1993, respectively. He had been with National Chung Cheng University in Taiwan from 1993 to 2011. He was the Deputy Director of Computer Center (1998–1999), the chairman of the Electrical Engineering Department (2003–2006) and the director of the Opti-mechatronics Institute of the University (2010–2011). He has been a member of IEEE since 1993 and a Fellow of the Institution of Engineering and Technology (FIET). His research interest includes methodologies and analysis for various intelligent robot systems, machine learning, embedded system design and ASIC design for robotic applications.

Hung-Shyuan Lin received the M.S. degree in electrical engineering from National Sun Yat-sen University, Kaohsiung, Taiwan, in 2016. He was a research assistant in the Intelligent Robotics Information System (IRIS) Laboratory, National Sun Yat-sen University. His research interests include machine learning, robotics, neural networks and embedded systems.