Information Sciences 520 (2020) 1–14

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

An end-to-end inverse reinforcement learning by a boosting approach with relative entropy

Tao Zhang[a], Ying Liu[a], Maxwell Hwang[b], Kao-Shing Hwang[c,*], ChunYan Ma[a], Jing Cheng[d]

[a] School of Software, Northwestern Polytechnical University, Shaanxi 710072, China
[b] School of Medicine, Zhejiang University, Hangzhou, China
[c] Department of Electrical Engineering, National Sun Yat-sen University, Kaohsiung 80424, Taiwan
[d] School of Computer Science and Engineering, Xi'an Technological University, Shaanxi 710021, China

ARTICLE INFO

Article history:
Received 17 October 2019
Revised 13 January 2020
Accepted 16 January 2020
Available online 17 January 2020

Keywords:
Inverse reinforcement learning
Imitation learning
State encoding
Adaboost
Relative entropy

ABSTRACT

Inverse reinforcement learning (IRL) involves imitating expert behaviors by recovering reward functions from demonstrations. This study proposes a model-free IRL algorithm to solve the dilemma of predicting the unknown reward function. The proposed end-to-end model comprises a dual structure of autoencoders in parallel. The model uses a state encoding method to reduce the computational complexity for high-dimensional environments and utilizes an Adaboost classifier to determine the difference between the predicted and demonstrated reward functions. Relative entropy is used as a metric to measure the difference between the demonstrated and the imitated behavior. The simulation experiments demonstrate the effectiveness of the proposed method in terms of the number of iterations that are required for the estimation.

1. Introduction

Manually programming the behavior of robots is time-consuming and difficult. For experts, demonstrating the desired behavior is easier than manually programming. The process of learning from demonstrations is called imitation learning [4,20], whereby only samples of trajectories are provided to learners.

A Markov Decision Process (MDP) [7,19] is eminently suited for reinforcement learning (RL) [26]. However, for some complex tasks or environments, manually adjusting the reward function is a generically expensive and time-consuming process, which limits the applicability of RL. If an agent is designed for desired behaviors, inverse reinforcement learning [2,17] allows the reward function to be adjusted, instead of specifying a bespoke reward function. Hence, no explicit reward function is founded. IRL also provides a framework to estimate the reward function for determining an optimal policy. However, IRL methods encounter challenges in practical applications. One major problem involves learning an unknown reward function that best supports a given policy or observed behavior.

Many IRL models have been proposed in recent years. These models are classified as model-based or model-free IRL. Most are model-based and assume that the expert is optimally acting within a MDP or that the behavior can be

* Corresponding author.
E-mail address: hwang@ccu.edu.tw (K.-S. Hwang).

https://doi.org/10.1016/j.ins.2020.01.023
0020-0255/

accurately learned using the demonstrated trajectories. Abbeel and Ng [1] determined an unknown reward function using inverse reinforcement learning by assuming that some vectors for the features [27,29] and that the reward function can be expressed as a linear combination of these features. However, it is difficult to update the reward function and the learning policy. Choi and Kim [5] extended this approach by building logical conjunctions of those component features that are relevant to the sample policy. To allow potentially complex situations, nonlinear reward functions [14,23] and multiple reward functions [10] have also been used. These functions respectively represent the reward function as a nonlinear combination and a nonparametric Bayesian problem with multiple reward functions. However, determining a solution using these methods is considerably complex. To use IRL in large state space, a Bellman Optimality Equation was used in a previous study [16]. However, this study assumed that the transition model and the action set remain unchanged for the subject. In recent years, entropy has been used to resolve the ambiguity in choosing a distribution over decisions. Ziebart et al. [30] proposed Maximum Entropy Inverse Reinforcement Learning (MaxEnt IRL), and based on this work, Maximum deep Entropy Inverse Reinforcement Learning (DeepEnt IRL) was proposed [25]. Although DeepEnt IRL is sufficient for real robots, its limitations are that it repeatedly solves the MDP and assumes that the dynamics are known.

Model-free IRL algorithms have been proposed to address the problem that assumptions are often unsatisfied in a large, continuous state space. Boularias et al. [3] proposed a relative entropy IRL method inspired by MaxEnt IRL [30]. Relative entropy was used to measure the gap between the empirical distribution of the state-action trajectories and the learned policy [28], which is minimized by stochastic gradient descent. However, the features must be selected manually, which is difficult for real application. Choi et al. [6] proposed a model-free density-based IRL algorithm that does not require model dynamics. The method first computes the empirical state action distribution for an expert and then determines the reward function that matches the density function. However, the features for these methods are high-dimensional, which is not suitable for complex tasks. Herman et al. [11] presented a gradient-based IRL approach that simultaneously estimates the system's dynamics. The bias of the demonstrations is taken into account, but only the fully observed MDP case was studied. To address the need for informative features and effective regularization, Finn et al. [8] presented an algorithm that learns arbitrary nonlinear cost functions. To address the difficulty of identifying the cost function for unknown dynamics, an efficient sample-based approximation was formulated. However, when pre-training is complex, this method limits further learning of the reward function, which can create other problems. Most of these methods also often use sampling to compute the reward function, which is time-consuming.

In this study, the dilemma of predicting the unknown reward function in IRL is solved. This study uses Adaboost to increase the discrimination capability for IRL and to adapt the weights of the reward function, which is a linear combination of weak classifiers [13]. Relative entropy is used to determine the difference between an expert's trajectory and an agent's imitation of this trajectory. The proposed IRL method minimizes the relative entropy between demonstrated behaviors and predicted trajectories. The proposed method, which is named Ada-entropy IRL, uses a state encoding method to simplify the definition and extraction of state space. An Adaboost classifier is used to determine the optimal reward function. The proposed method estimates an implicit reward function by comparing the trajectories between the behaviors to allow imitation learning in large, complex environments.

The main contributions of the paper can be concluded as:

(1) A novel model-free IRL is introduced for end-to-end scenarios [24] where images of the demonstrations are observed to directly derive the predictions of the reward function.

(2) The proposed Ada-entropy algorithm simply resolves the sub-gradients of the min-max problem in IRL [18,30] accurately so that imitation learning can be achieved.

(3) The proposed method allows learning in complex environments because an autoencoder is used for state encoding for feature representations of the environment so as to avoid the curse of dimensionality problem.

The remainder of this paper is organized as follows: Section 2 introduces the framework for the Adaboost, and reviews Maximum Entropy IRL and Maximum Entropy Deep IRL, both of which use a similar approach to the proposed method. Section 3 uses the proposed Ada-entropy IRL method to determine the reward function. Section 4 describes the simulated experiments. The results demonstrate the effectiveness and superiority of the Ada-entropy IRL algorithm. Finally, Section 5 draws conclusions.

2. Preliminaries

2.1. Adaboost classifier

Adaboost is a boosting-based classification algorithm. An Adaboost classifier is a strong classifier comprising multiple basic classifiers with different adaptive weights. The weight of each basic classifier is adjusted, depending on the error rate for data classification to allow adaptive classification for a limited class of samples. In each iteration, it calls a base learner that returns a classifier, and assigns a weight coefficient to it. The final classification is decided using a weighted "vote" by the base classifiers. The smaller the error in the base classifier, the greater is its weight in the final vote.

For a set of two classifier training data $T = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$, where $x_i \in X \subseteq R^n$, hence, each x_i belongs to a domain or instance space X and each label y_i belongs to a label set Y, thus $Y = \{-1, +1\}$ is assumed.

The error rate for a basic classifier $G_m(x)$ is denoted as e_m. Error rate is the ratio of incorrect classified training data, which is defined as:

$$\begin{aligned} e_m &= \sum_{i=1}^{N} P(G_m(x_i) \neq y_i) \\ &= \omega_{mi} I(G_m(x_i) \neq y_i) \\ &= \sum_{G_m(x_i) \neq y_i} \omega_{mi} \end{aligned} \tag{1}$$

where ω_{mi} denotes the weight of sample i in the mth iteration, and $\sum_{i=1}^{N} \omega_{mi} = 1$. $P(G_m(x_i) \neq y_i)$ is the probability of $G_m(x_i) \neq y_i$. The indicator function is denoted as I, when $G_m(x_i) \neq y_i$, $I(G_m(x_i) \neq y_i) = 1$ and vice versa.

The weight distribution D_{m+1} and the sample weight $\omega_{m+1,i}$ are then updated as:

$$D_{m+1} = (\omega_{m+1,1}, \ldots, \omega_{m+1,i}, \ldots, \omega_{m+1,N}) \tag{2}$$

$$\begin{aligned} \omega_{m+1,i} &= \frac{\omega_{mi}}{Z_m} \exp(-\alpha_m y_i G_m(x_i)) \\ &= \begin{cases} \frac{\omega_{mi}}{Z_m} e^{-\alpha m}, G_m(x_i) = y_i \\ \frac{\omega_{mi}}{Z_m} e^{\alpha m}, G_m(x_i) \neq y_i \end{cases} \end{aligned} \tag{3}$$

where the coefficient of the basic classifier $G_m(x)$ is denoted as α_m, and the normalization factor is Z_m :

$$Z_m = \sum_{i=1}^{N} \omega_{mi} \exp(-\alpha_m y_i G_m(x_i)) \tag{4}$$

Finally, the linear combination of the basic classifier is constructed as:

$$f(x) = \sum_{m=1}^{M} \alpha_m G_m(x) \tag{5}$$

The output G(x) is the final strong classifier, which is a combination of weighted basic classifiers.

$$G(x) = sign(f(x)) = sign(\sum_{m=1}^{M} \alpha_m G_m(x)) \tag{6}$$

2.2. Maximum entropy inverse reinforcement learning

To address ambiguity in a structured way, maximum entropy is utilized to match feature counts. Maximum entropy is the optimum issue, and the problem is transformed as:

$$\begin{aligned} &\max \; -p \log p \\ &s.t. \sum_{path \zeta_i} P(\zeta_i) f_{\zeta_i} = \widetilde{f} \\ &\sum P = 1 \end{aligned} \tag{7}$$

where f represents the feature expectation for the agent. Using a Lagrange multiplier, the problem is transformed as:

$$\min L = \sum_{\zeta_i} p \log p - \sum_{j=1}^{n} \lambda_j (p f_j - \widetilde{f}_j) - \lambda_0 (\sum p - 1) \tag{8}$$

After a differential operation, the probability of maximum entropy is calculated as:

$$p = \frac{\exp(\sum_{j=1}^{n} \lambda_j f_j)}{\exp(1 - \lambda_0)} = \frac{1}{Z} \exp(\sum_{j=1}^{n} \lambda_j f_j) \tag{9}$$

where the parameter λ_j is the parameter for the reward function, which is calculated using the Maximum Likelihood Method. However, the likelihood function for the demonstrations can be calculated only when the transition function T is known. Although relative entropy resolves the ambiguity in choosing a distribution over a decision, applying in real applications is difficult. During the process of learning the reward function, the features must be selected manually.

2.3. Maximum entropy deep inverse reinforcement learning

For Maximum Entropy Deep IRL, a network architecture is used to approximate the reward function. The state reward is defined as:

$$\begin{aligned} r &\approx g(f, \theta_1, \theta_2, \ldots, \theta_n) \\ &= g_1(g_2(\cdots(g_n(f, \theta_n), \ldots), \theta_2, \theta_1) \end{aligned} \tag{10}$$

where the network parameter is denoted as $\theta_{1,2,\ldots,n}$.

The problem is then defined as maximizing the joint posterior distribution of the observed expert demonstrations D:

$$L(\theta) = logP(D, \theta|r) = \underbrace{logP(D, \theta|r)}_{LD} + \underbrace{logP(D, \theta|r)}_{L\theta} \tag{11}$$

The complete gradient is calculated as:

$$\frac{\partial L}{\partial \theta} = \frac{\partial L_D}{\partial \theta} + \frac{\partial L_\theta}{\partial \theta} \tag{12}$$

The expected value is defined as:

$$E[\mu] = \sum_{\varsigma:\{s,a\}\in\varsigma} P(\varsigma|r) \tag{13}$$

However, when a wider convolutional filter is used for Maximum Entropy Deep IRL, the demand for expert demonstrations is increased. In these model-based methods, the MDP must be repeatedly solved, and the dynamics are assumed to be known.

To address the above problems, this study proposes a novel model-free method for inverse reinforcement learning, whereby the feature vector is encoded to simplify the state space. An Adaboost classifier with a set of basic classifiers that have different adaptive weights is used to adjust the weight of the reward. Adjusting the weight of the reward function minimizes the relative entropy between the expert's feature vector and the agent's feature vector.

3. The proposed method

The proposed Ada-entropy IRL algorithm uses an Adaboost algorithm and relative entropy to define a reward function that allows so-called end-to-end inverse reinforcement learning. The proposed algorithm calculates the combination of parameters for the relative entropy between the sampled sequential training images and the agent's using the Adaboost method. The conceptual end-to-end structure of the proposed method is shown in Fig. 1. The input is a screenshot image of the demonstration and the imitated trajectory and the output of the proposed in-parallel dual structure of autoencoders is the predicted reward function. The dual model accommodates the sequential images of the demonstrations and the current trajectory, each pair of which is handled respectively, by the corresponding autoencoder pair between the demonstration and the imitation for feature representations.

Fig. 1 presents how the proposed method works during the mth iteration. For each action in a sequence that is denoted by the numbered boxes in the trajectories in the upper and lower part of the figure, a feature is encoded by an autoencoder

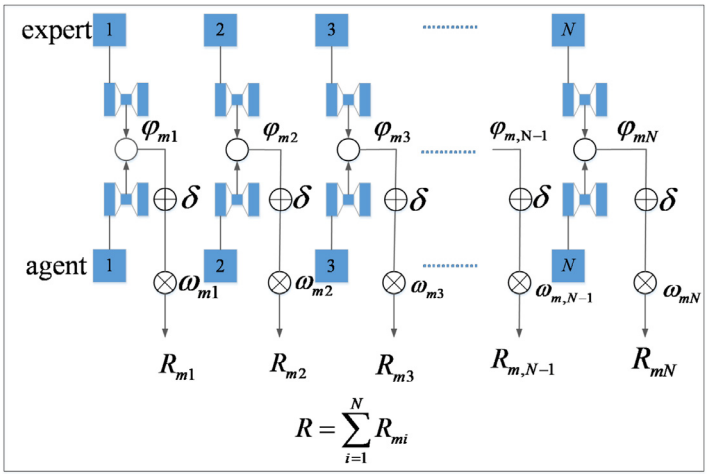


Fig. 1. The architecture of the proposed Ada-entropy IRL method; in this illustrative example, the mth iteration consists of N frames (actions) which are used for calculating of the combination of the reward function $R = -\omega(\varphi + \delta) = \sum_{i=1}^{N} R_{mi} = \sum_{i=1}^{N} -\omega_{mi}(\varphi_{mi} + \delta)$.

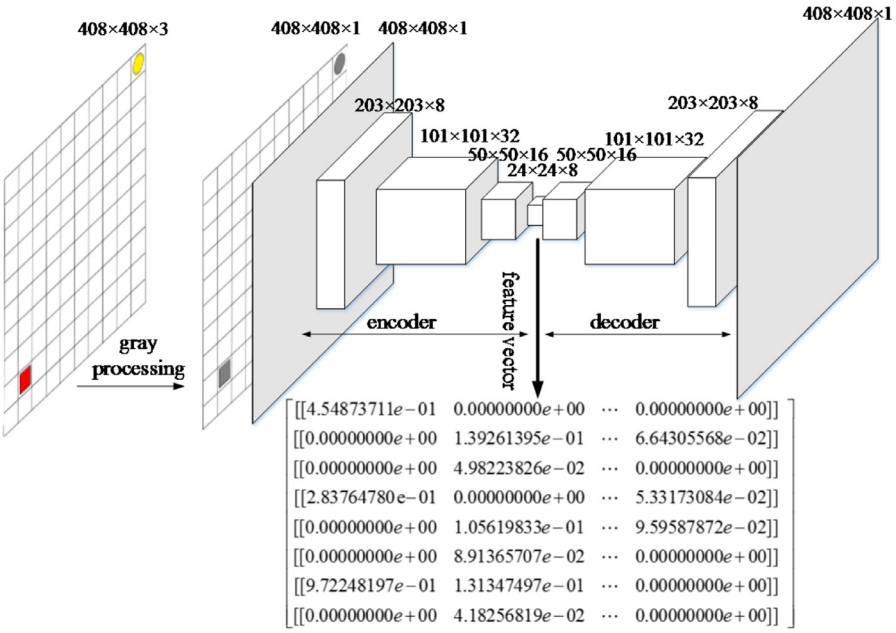


Fig. 2. An example of state encoding.

and the difference between those for the demonstration and imitation is defined as the baseline φ_{mi}. Since in an extreme case, the reward function R_{mi} is null when all $\varphi_{mi} = 0$, such a negative small scalar bias δ is introduced. The added results multiplies with the reward weight ω_{mi}, the final output R_{mi} is attainable, which is explained in Eq. (23) in detail.

3.1. State encoding

Most previous studies manually or empirically selected the designed feature vectors, such as feature expectation [1], which produces a state space that is massive and complex. The definition of features is also ambiguous. When a task is decomposed into lower-level behaviors, the dimensionality of the state space becomes so extensive that it inhibits the execution of IRL.

This study uses the state-encoding method to discretize and simplify the state space [12]. An autoencoder is used to simplify the high dimension state space. An autoencoder [15,21] is composed of two main parts: an encoder and a decoder, which maps input into the code and maps the code to reconstruct the original input respectively. This study uses only the encoder part to simplify the state space. The input for the end-to-end model is the screenshot of the trajectories and the output is the predicted reward function. The characteristics of each action are expressed in the form of an image to a vector and the dimension of the feature vectors is significantly reduced. An example of state encoding is shown in Fig. 2.

Fig. 2 displays the extraction of the feature vector for a simulation screenshot in a maze environment using an autoencoder. The screenshot firstly undergoes gray processing to decrease the number of features. After encoding, the middle hidden layer outputs the feature vector for this action, whose dimension significantly decreased compared with the input. Feature vector extraction allows further comparison and evaluation of imitation learning.

3.2. The weights of basic classifier

When certain rewards constantly misleads an agent to learn the unexpected policy, the value of the reward function should be changed with a large ratio. That is, the larger the error rate is, the larger ratio the value changes. The thought corresponds to the Adaboost algorithm, and the concept of adjusting the weight value of AdaBoost is used on the scale (reward weight) that each element must be updated.

The proposed method represents the comparison of trajectories in IRL as a classification problem to allow a fast and flexible approach to adjust the weight of the approximated reward function. After $m-1$ iterations, using Eq. (5), the basic classifier is expressed as:

$$f_{m-1}(x) = f_{m-2}(x) + \alpha_{m-1}G_{m-1}(x) \tag{14}$$

Then $f_m(x)$ is obtained from Eq. (14):

$$f_m(x) = f_{m-1}(x) + \alpha_m G_m(x) \tag{15}$$

To minimize the exponential loss of training datasets in the highest rate, α_m and $G_m(x)$ are given as:

$$\begin{aligned}(\alpha_m, G_m(x)) &= arg\min_{\alpha,G}\sum_{i=1}^{N}\exp[-y_i(f_{m-1}(x_i)+\alpha G(x_i))]\\ &= arg\min_{\alpha,G}\sum_{i=1}^{N}\overline{\omega}_{mi}\exp[-y_i\alpha G(x_i)]\end{aligned} \tag{16}$$

where $\overline{\omega}_{mi}=\exp[-y_i f_{m-1}(x_i)]$. Given that $\overline{\omega}_{mi}$ is independent of α_m and $G_m(x)$, it is not considered for minimality.

α_m and $G_m(x)$ are respectively denoted as α_m^* and $G_m^*(x)$ to minimize Eq. (16). For any $\alpha > 0$, $G_m^*(x)$ is calculated as:

$$G_m^*(x) = \arg\min_{G}\sum_{m=1}^{N}\overline{\omega}_{mi}I(y_i \neq G(x_i)) \tag{17}$$

The classifier $G_m^*(x)$ is the basic classifier $G_m(x)$ for Adaboost because it minimizes the error rate for training data in the m^{th} iteration.

According to Eq. (3), in Eq. (16):

$$\begin{aligned}\sum_{i=1}^{N}\overline{\omega}_{mi}\exp[-y_i\alpha G(x_i)] &= \sum_{y_i=G_m(x_i)}\overline{\omega}_{mi}e^{-\alpha}+\sum_{y_i\neq G_m(x_i)}\overline{\omega}_{mi}e^{\alpha}\\ &= (e^{\alpha}-e^{-\alpha})\sum_{i=1}^{N}\overline{\omega}_{mi}I(y_i\neq G(x_i))+e^{-\alpha}\sum_{i=1}^{N}\overline{\omega}_{mi}\end{aligned} \tag{18}$$

Substituting $G_m^*(x)$ into Eq. (18) for α equal to 0, the result is:

$$\alpha_m^* = \frac{1}{2}\log\frac{1-e_m}{e_m} \tag{19}$$

where α_m represents the importance of the basic classifier $G_m(x)$ in the final strong classifier. If $e_m \leq \frac{1}{2}$, $\alpha_m \geq 0$, and α_m is inversely proportional to e_m, then the lower the error rate, the more important the basic classifier is in the final strong classifier. That is, for the combination of basic classifiers, the lower the error rate for the basic classifier, the greater is its role in the vote.

3.3. Obtaining reward using relative entropy

For IRL, given an MDP, the unknown reward function $R(s)$ is defined as:

$$R(s) = \omega^T\phi(s) \tag{20}$$

where $\phi(s)$ is a vector of features: $S \rightarrow [0, 1]^k$, and $\omega \in \mathrm{R}^k$.

For an Ada-entropy IRL algorithm, the repeatedly estimated reward $R(s)$ varies over iterations in accordance with the difference between the demonstrated and the imitated trajectories. Therefore, relative entropy [9] is used to measure similarity. The higher the relative entropy $KL(agent||expert)_{mi}$, the worse is the imitation and the agent receives a negative reward, and vice versa.

The relative entropy $KL(agent||expert)_{mi}$ is defined as:

$$KL(agent||expert)_{mi} = \sum_{i=1}^{N}a_i\log\frac{a_i}{b_i} \tag{21}$$

where a_i belongs to the feature vector for the agent $\phi_{mi}=\{a_1, a_2, \ldots, a_N\}$ and b_i belongs to the feature vector for the expert $\phi'_{mi}=\{b_1, b_2, \ldots, b_N\}$.

Eq. (21) measures the difference between the two trajectories in terms of the relative entropy between the feature vectors. $KL(agent||expert)_{mi}$ is denoted as φ_{mi}, which is used as the base to update the expected reward function. The reward function R_{mi} for the sample i in the m^{th} iteration is updated as the linear combination of ω_{mi}, φ_{mi}:

$$R_{mi} = -\omega_{mi}\cdot(\varphi_{mi}+\delta) \tag{22}$$

and for the whole trajectory in the vector form is:

$$R = -\omega(\varphi+\delta) = \sum_{i=1}^{N}\mathrm{R}_{mi} = \sum_{i=1}^{N}-\omega_{mi}(\varphi_{mi}+\delta) \tag{23}$$

Then Q-Learning for neural networks, such as Neural Fitted Q Iteration (NFQ) [22], is employed to compute an optimal policy π_{i} using the current predicted reward function. If the imitated trajectory is the same as the expert trajectory in sample k, then $\varphi_{mi}=0$, and vice versa. δ is a negative small value that prevents reward R_{mi} from being equal to 0 when $\varphi_{mi}=0$. If the agent fails to emulate the behavior of the expert, then the agent receives a negative reward. If the imitated behavior the is same as the demonstration, then the agent receives a positive reward, which is close to 0. The objective

of inverse reinforcement learning is to minimize the distance between the feature expectations of the expert and the feature expectation of the agent. IRL determines a reward function that produces the trajectory that is closest to the expert's demonstration. The agent obtains a maximum reward and avoids previous errors.

φ_{mi} is the basis of the linear combination of the approximated reward function and is used to determine an appropriate weight for the reward. The weight of the reward $\omega_{m+1,i}$ is updated as:

$$\omega_{m+1,i} = \omega_i + D_{m+1,i} \cdot \varphi_{m+1,i} \cdot \gamma^i \tag{24}$$

where γ is the discount factor that ensures the convergence of the value of ω_{mi}. The weight of the reward ω_{mi} is incrementally updated over many iterations. Therefore, if the imitated behavior is inaccurate after a long period, then the weight of the reward ω_{mi} increases over those iterations, the incorrect classified dataset receives additional attention for the next iteration and the agent receives a negative reward with a large absolute value. The propsed Ada-entropy IRL algorithm is shown in Algorithm 1.

Algorithm 1 Ada-entropy IRL.

Step 1: Initializing
- 1.1 $\omega_{1,i}$ randomly with non-zero value for all $s \in S$.
- 1.2 $D_{0i}=\frac{1}{M}$, M is the number of the features.
- 1.3 $i \leftarrow 0$, $m \leftarrow 0$.

Step 2: Obtaining feature vector ϕ'_{mi} for the expert's indexed action image frame from the autoencoder.

Step 3: Repeat (for each iteration):
- 3.1 $m \leftarrow m+1$
- 3.2 $i \leftarrow i+1$
- 3.3 $e_m \leftarrow \sum_{\varphi_{mi} \neq 0} D_{mi}$
- 3.4 $\alpha_m = \frac{1}{2} ln \frac{1-e_m}{e_m}$
- 3.5 $D_{m+1,i} = D_{mi} \cdot \begin{cases} 1 & \text{if} \varphi_{mi} \neq 0 \\ e^{-\alpha_m} & \text{if} \varphi_{mi} = 0 \end{cases}$
- 3.6 Normalizing $D_{m+1,i}$
- 3.7 $\omega_{m+1,i} = \omega_i + D_{m+1,i} \cdot \varphi_{m,i+1} \cdot \gamma^i$
- 3.8 Obtaining reward $R_{mi} = -\omega_{mi} \cdot (\varphi_{mi} + \delta)$
- 3.9 Using Q-Learning for neural networks, such as Neural Fitted Q Iteration (NFQ) [22], to compute an optimal policy π_i employing the current predicted reward function.
- 3.10 Obtaining feature vector of agent ϕ_{mi}
- 3.11 Computing the relative entropy $\varphi_{m+1,i}$ according to Eq. (21)
- 3.12 Until $\sum_{i=1}^{N} \varphi_{mi} = 0$

There are two significant advantages of the Ada-entropy IRL algorithm: the agent does not need to use an MDP model; and the proposed IRL method is applicable to an end-to-end RL scenario. The simulation results for the proposed method are shown in the next section, and the proposed algorithm is detailed below.

4. Simulations

Three sets of simulations are used to determine the performance of the Ada-entropy IRL. The first domain is a maze world, the second is a car simulation, and the third uses the Atari game, Flappy Bird. The scenarios vary from simple to complicated. The performance of the three different common IRL methods - maximum entropy IRL (MaxEnt) [30], maximum entropy deep IRL (DeepEnt) [25], and function approximation IRL with a neural network (FunNN) [16] - are determined by comparing the reward map. The vertical dimension of the reward map is the Y label, and the horizontal dimension is the X label of the position. The depth of color for each position represents the reward for each position. If the color is lighter, then the reward is greater.

4.1. Maze environment

Fig. 3 shows a maze with a size of 10 grids × 10 grids that is used for the simulation. The rectangular red spot is the starting point, and the goal is the yellow circle in the upper-left corner of the map. The reward function is unknown to the agent. To reach the goal in the maze, the trajectory that is demonstrated by an expert is sampled. This example uses a twenty-step demonstration trajectory. The agent can perform four actions: up, down, left and right. If it hits the wall, then the session ends. The learning rate is 0.8, and the exploration rate ε=0.1. To avoid a local optimum, the discount factor γ is set to 0.8, and the bias value δ is -0.0001.

The reward map after 300 iterations is illustrated in Fig. 4. Fig. 4(a) shows the trajectory of the expert. Figs. 4(b), (c), (d) and (e) respectively show the reward for Ada-entropy IRL, MaxEnt, DeepEnt and FunNN respectively.

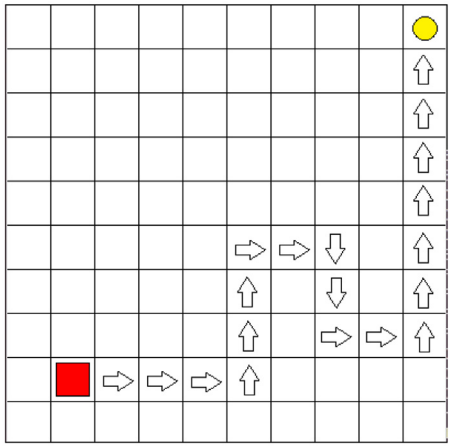
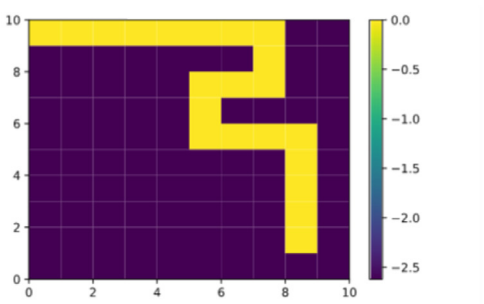
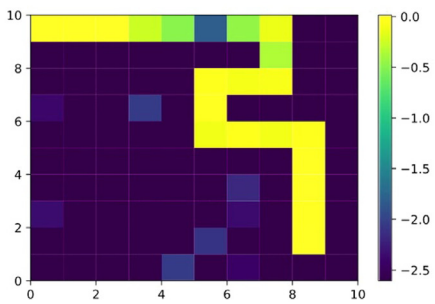


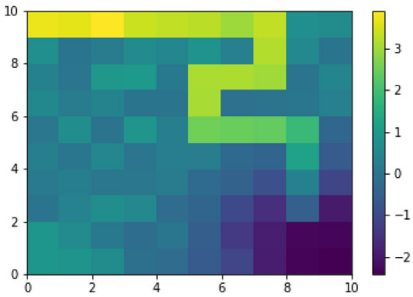
Fig. 3. The expert's demonstration for the maze. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)



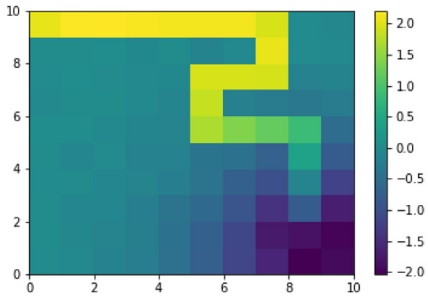
(a) The trajectory demonstrated by an expert



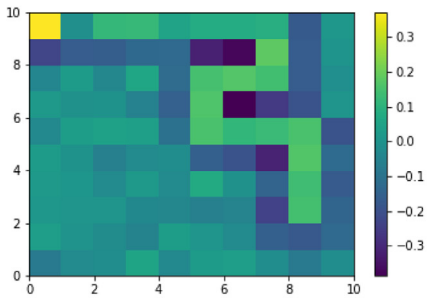
(b) The trajectory predicted by Ada-entropy IRL



(c) The trajectory predicted by MaxEnt IRL



(d) The trajectory predicted by DeepEnt IRL



(e) The trajectory predicted by FunNN IRL

Fig. 4. Trajectories in the maze: The color of each grid represents the reward for each position. The vertical bar on the right-hand side is the scale for the reward values. A lighter color represents a greater reward.

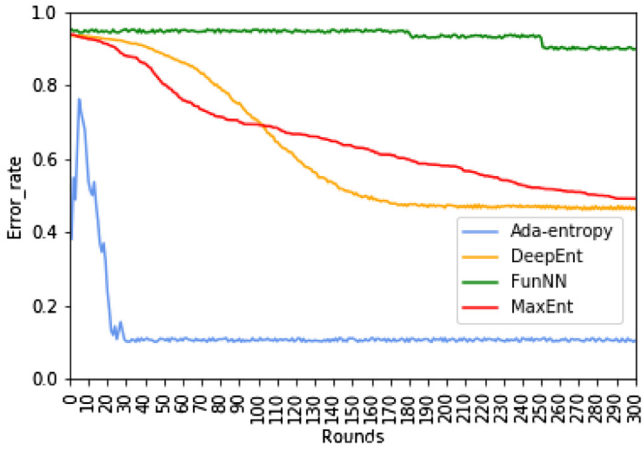


Fig. 5. Error rate for a maze.

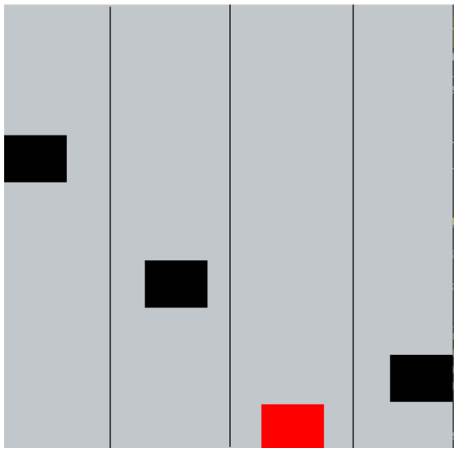


Fig. 6. Car simulation. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

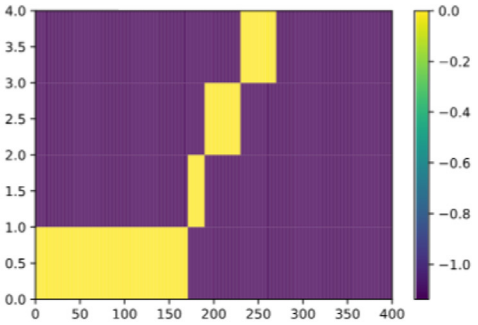
All of the algorithms produce trajectories that are similar to the demonstration, but the trajectory of the proposed Ada-entropy method is clear. Additional tests are performed to compare the results of imitation learning by using these methods. If the action of the agent is the same as that of the expert at the same time step, the action is deemed to be correct, and vice versa. Fig. 5 displays shows the error rate for imitation learning. The error rate is the incorrect action accounts for the proportion of all actions.

The proposed method significantly performs better than the others, in terms of learning speed and convergence. The average error rate for the Ada-entropy IRL drops to 0.1 in the 35th episode. The value for δ is set to 0.1, hence 10% randomness are observed in each action, and the error rate for the proposed method cannot reach 0. The error rate for MaxEnt IRL falls to 0.5 in the 290th episode and the error rate for DeepEnt IRL decreases to 0.5 in the 170th episode. The error rate for FunNN IRL decreases to 0.9 in the 250th episode. The following simulations involve additional complicated scenarios.

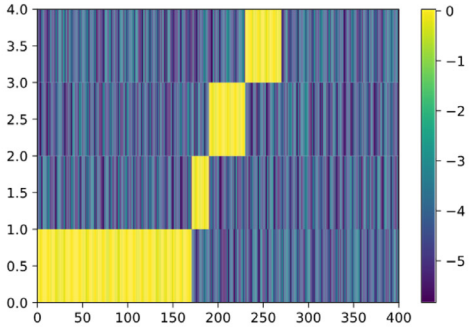
4.2. Car simulation

The second simulation uses a simplified car driving simulator. The four algorithms are used to simulate an expert's demonstration. Fig. 6 shows a screenshot for the simulation. The agent (the red rectangle) drives on a four-lane road. Other cars (black rectangles) drive in the opposite direction, hence the agent must take one of two actions: moving left or right to avoid a collision. The expert gives a 260-step demonstration for approximately one minute. If the agent's car fails to prevent accidents, then the new session starts again. The learning rate for this simulation is 0.8 and the discount rate is 0.98, including 10% for random actions. The discount factor γ for the Ada-entropy IRL algorithm is 0.8 and the bias value δ is -0.0001.

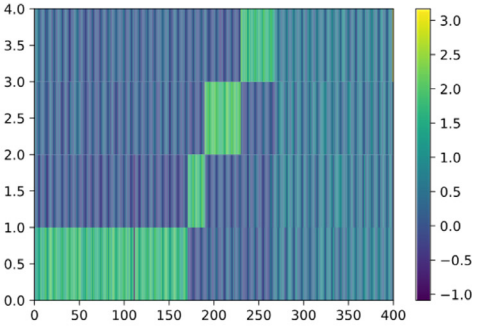
The expert trajectory is presented in Fig. 7(a). Fig. 7(b), (c) and –(d) respectively show the reward maps after 300 iterations of Ada-entropy IRL, MaxEnt, DeepEnt and FunNN respectively.



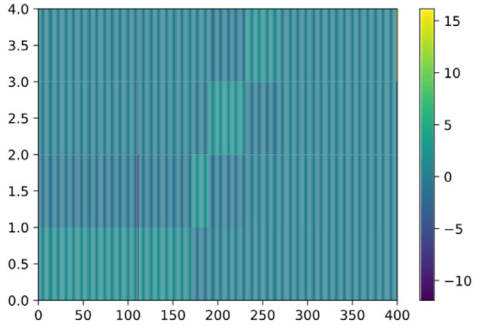
(a) The trajectory demonstrated by an expert



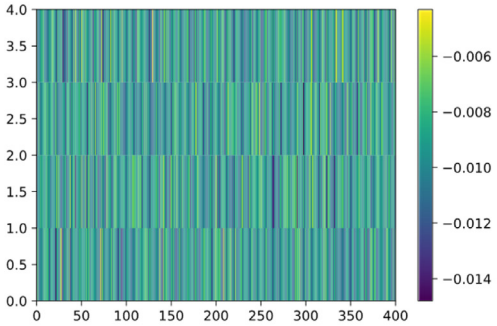
(b) The trajectory predicted by Ada-entropy IRL



(c) The trajectory predicted by MaxEnt IRL



(d) The trajectory predicted by DeepEnt IRL

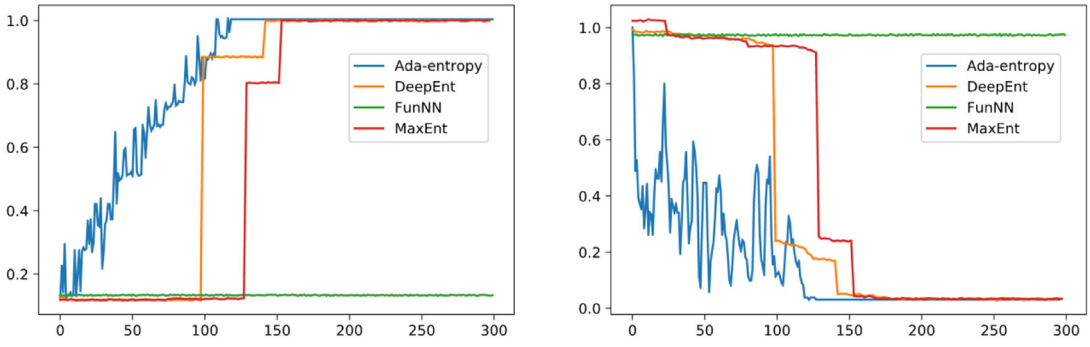


(e) The trajectory predicted by FunNN IRL

Fig. 7. Trajectories in car simulation: The x-label denotes the order of the sequence and the y-label denotes the position of agent: a value of 0–1 denotes that the agent is driving in the far left lane, whereas a value of 1–2 denotes that the agent is driving in the second lane from the left. The color of each grid represents the reward for each position. The vertical bar on the right-hand side is the scale for the reward values. A lighter color represents a greater reward.

In this simulation, the state space is large, hence the FunNN IRL with neural network is especially sensitive to the value of the MDP transition probability P, which is challenging to model. Therefore, updating the reward is difficult. For this simulation, agent moves left or right to avoid a crash. This state is defined as "survived". If the agent hits other cars, the state of the agent is defined as "dead". The survival rate is defined as the ratio of "survived" states to the number of expert demonstrations. The average survival rate and the error rate for these methods are respectively shown in Fig. 8(a) and 8(b). The x-axis indicates the number of iterations, and the y-axis indicates the survival rate or the error rate.

In a more complex environment, a more significant difference is observed in the results of these methods. High-dimensional environments render challenges for agents to learn using these methods because it is difficult to determine



(a) Survival rate for the car simulation

(b) Error rate for the car simulation

Fig. 8. Survival rate and error rate in car simulation.

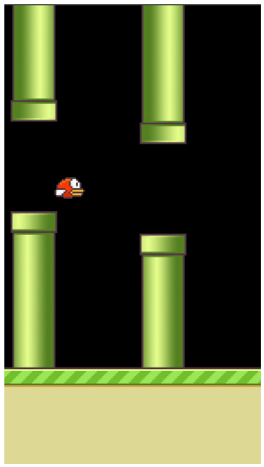
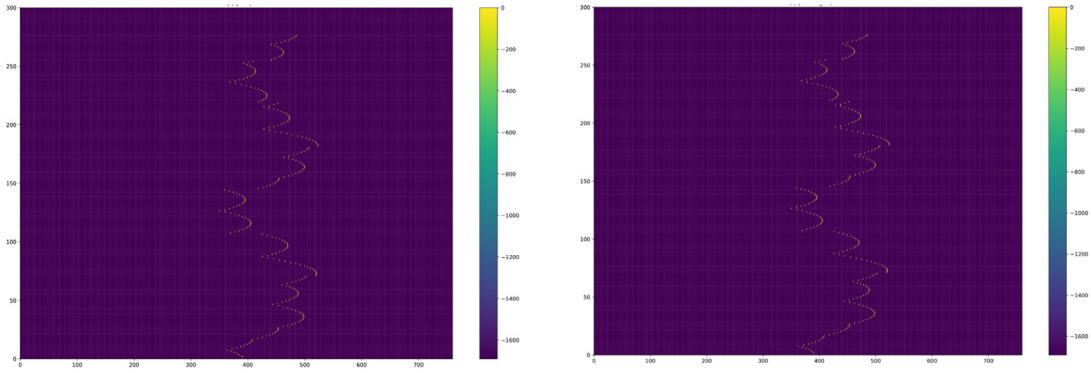


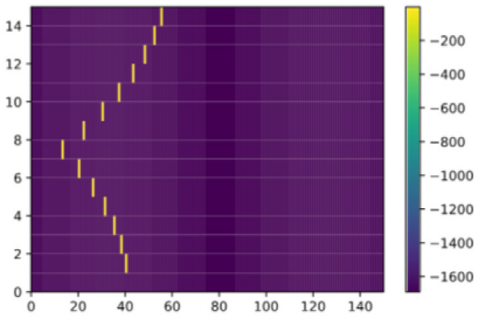
Fig. 9. Flappy Bird.



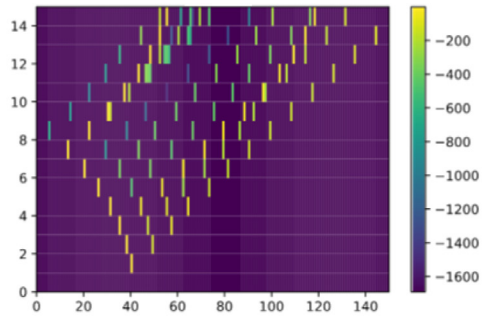
(a) The trajectory demonstrated by an expert

(b) The trajectory predicted by Ada-entropy IRL

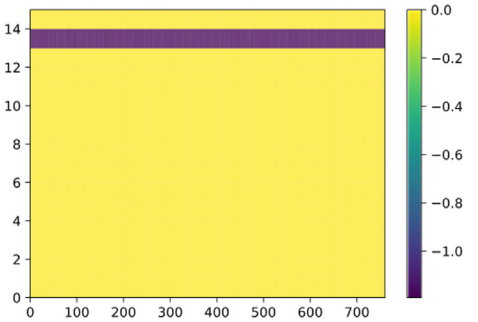
Fig. 10. Trajectories in Flappy Bird: The y-axis represents the time-step in the sequence and the x-axis represents the relative height of the bird. The color of each grid represents the reward for each position. The vertical bar on the right-hand side shows the scale for the reward values. A lighter color represents a greater reward.



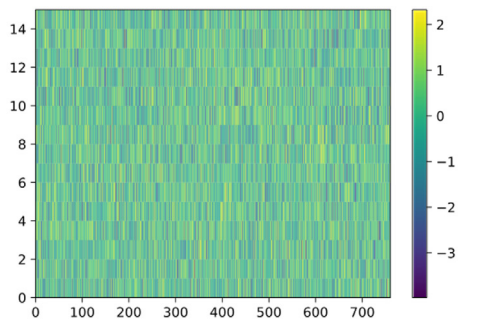
(a) The trajectory demonstrated by an expert



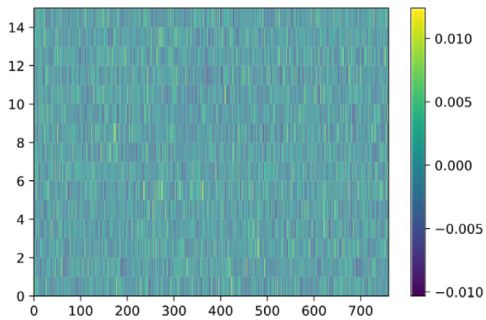
(b) The trajectory predicted by Ada-entropy IRL



(c) The trajectory predicted by MaxEnt IRL



(d) The trajectory predicted by DeepEnt IRL



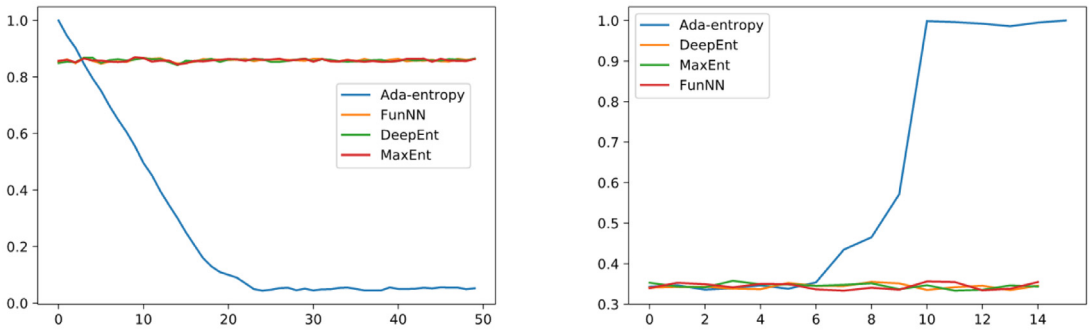
(e) The trajectory predicted by FunNN IRL

Fig. 11. Trajectories in Flappy Bird for 15 steps.

the transition probability for these model-based methods. The FunNN IRL is most sensitive to the value of P. Therefore, the gradient barely descends in this environment, and this method entails the worst results. A comparison of the results in Fig. 8 shows that the proposed method converges at approximately the 120th iteration and the two other methods respectively converge at the 150th and 160th iterations. The Ada-entropy IRL learns and converges faster than the three other methods.

4.3. Flappy bird

The Atari game, FlappyBird, involves a bird flying forward at a constant speed and the game players control the flight's altitude. If the player relaxes his or her fingers, then the bird rapidly descends under the influence of gravity. Thus the player must maintain the bird's height, and avoid random pillars. If the bird hits the pillars, then it dies and a new round



(a) Error rate for Flappy Bird (b) Survival rate for Flappy Bird

Fig. 12. Survival rate and error rate in Flappy Bird.

begins. The bird moves at high speed and the distance between pillars varies randomly. Hence, attaining a high score is difficult. The interface of the game is shown in Fig. 9. To allow the game to be automatically played by the agent, one minute's demonstration involving 280 actions is given by the expert for demonstrating how to avoid pillars. The learning rate is 0.8, the discount rate is 0.98 and the bias value δ is -0.0001. For each movement, 10% randomness are observed to avoid becoming trapped in a local optimum.

The 2D reward map after 300 iterations is displayed in Fig. 10. The demonstration by the expert is shown in Fig. 10(a) and the imitated trajectory using the proposed method is exhibited in Fig. 10(b). The reward for the proposed method is very close to the trajectory demonstrated by the expert.

The state space for this simulation is extremely vast. Each demonstration involves 150 states and a total of 230 successive actions results in 34,500 state-action pairs. The state space is extensively large for the three algorithms that are compared: MaxEnt IRL, DeepEnt IRL and FunNN IRL. To simplify the state space, the first 15-step demonstrations are selected as the expert's trajectory. A comparison of the reward for the first 15 steps after 50 iterations of these algorithms is exhibited in Fig. 11(a), (b), (c), (d) and –(e) respectively show the reward for the demonstration, Ada-entropy IRL, MaxEnt, DeepEnt and FunNN. Where the y-axis represents the time-step in the sequence and the x-axis represents the relative height of bird.

In this very complex environment, updating the reward function for the three other methods is tough. The average error rates and the survival rates are compared in Fig. 12(a) and (b).

The environment is extremely large and complex. Therefore, it is challenging to determine a value for the MDP transition probability, especially if the flight path describes a parabola. The methods that are compared poorly perform in terms of the survival rate and the error rate, which do not converge for this complex environment. The proposed method is applicable in a large and high dimension state space.

5. Conclusions

This study proposes a model-free IRL to allow imitation learning, without requiring an MDP model or bespoke selection of features. The Ada-entropy IRL algorithm is an end-to-end approach that uses a state encoding method to simplify the problem for complex and high-dimensional environments. An Adaboost classifier is used as a two-class classifier to adaptively assign the weight for a reward. Relative entropy is employed to measure the difference between the demonstrated and imitated trajectories. Using the approximated reward function, an agent learns the policy to perform behaviors that are similar to those demonstrated by an expert. The experimental results show the effectiveness and superiority of this method, especially for complex environments, where the methods that are compared work poorly. Future studies will determine how to assign multiple reward functions for various demonstrations in more complex tasks.

Declaration of Competing Interest

None.

CRediT authorship contribution statement

Tao Zhang: Conceptualization, Methodology. **Ying Liu:** Formal analysis, Writing - original draft. **Maxwell Hwang:** Writing - review & editing. **Kao-Shing Hwang:** Supervision. **ChunYan Ma:** Software, Data curation. **Jing Cheng:** Software.

Acknowledgments

This work was partially supported by the Education Department of Shaanxi Province (18JK0380), Natural Science Basic Research Program of Shaanxi Province (2019JM-484), Graduate Creative Innovation Seed Fund of Northwestern Polytechnical University and State and Provincial Joint Engineering Lab. of Advanced Network, Monitoring and Control (GSYSJ2018005).

References

[1] P. Abbeel, A.Y. Ng, Apprenticeship learning via inverse reinforcement learning, in: Proceedings of the twenty-first international conference on Machine learning, ACM, 2004, doi:10.1145/1015330.1015430. 1–8
[2] S. Arora, P. Doshi, A Survey of Inverse Reinforcement Learning: Challenges, Methods and Progress, 2018, arXiv:1806.06877.
[3] A. Boularias, J. Kober, J. Peters, Relative entropy inverse reinforcement learning, in: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, 2011, pp. 182–189.
[4] X. Cai, Y. Ding, Y. Jiang, Z. Zhou, Expert-Level Atari Imitation Learning From Demonstrations Only, 2019, arXiv:1909.03773.
[5] J. Choi, K.E. Kim, Nonparametric Bayesian inverse reinforcement learning for multiple reward functions, in: Proceedings of the Advances in Neural Information Processing Systems, 2012, pp. 305–313.
[6] S. Choi, K. Lee, H.A. Park, S. Oh, Density Matching Reward Learning, 2016, arXiv:1608.03694.
[7] T. Dick, A. György, C. Szepesvári, Online learning in Markov decision processes with changing cost sequences, in: Proceedings of the International Conference on Machine Learning, 2014, pp. 512–520.
[8] C. Finn, S. Levine, P. Abbeel, Guided cost learning: deep inverse optimal control via policy optimization, in: Proceedings of the International Conference on Machine Learning, 2016, pp. 49–58.
[9] D.K. Ghasedi, H. Amirhossein, C. Deng, W. Cai, H. Huang, Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 5736–5745, doi:10.1109/iccv.2017.612.
[10] A. Gleave, O. Habryka, Multi-Task Maximum Entropy Inverse Reinforcement Learning, 2018, arXiv:1805.08882.
[11] M. Herman, T. Gindele, J. Wagner, F. Schmitt, W. Burgard, Inverse reinforcement learning with simultaneous estimation of rewards and dynamics, in: Proceedings of the Artificial Intelligence and Statistics, 2016, pp. 102–110.
[12] C. Hong, J. Yu, J. Zhang, X. Jin, K.H. Lee, Multi-modal face pose estimation with multi-task manifold deep learning, IEEE Trans. Ind. Inform. 15 (7) (2018) 3952–3961.
[13] W. Lee, C.-H. Jun, J.S. Lee, Instance categorization by support vector machines to adjust weights in adaboost for imbalanced data classification, Inf. Sci. 381 (2017) 92–103.
[14] S. Levine, Z. Popović, V. Koltun, Nonlinear inverse reinforcement learning with Gaussian processes, in: Proceedings of the Advances in Neural Information Processing Systems, 2011, pp. 19–27.
[15] Y. Li, Q. Pan, S. Wang, H. Peng, T. Yang, E. Cambria, Disentangled variational auto-encoder for semi-supervised learning, Inf. Sci. 482 (2019) 73–85.
[16] K. Li, M. Rath, J.W. Burdick, Inverse reinforcement learning via function approximation for clinical motion analysis, in: Proceedings of the IEEE International Conference on Robotics and Automation, IEEE, 2018, pp. 610–617, doi:10.1109/icra.2018.8460563.
[17] J. Lin, K.-S. Hwang, W. Pan, H. Shi, An ensemble method for inverse reinforcement learning, Inf. Sci. 512 (2020) 518–532.
[18] A.Y. Ng, S.J. Russell, Algorithms for inverse reinforcement learning, in: Proceedings of the International Conference on Machine Learning, 2000, pp. 2–9.
[19] F. Norman, P. Prakash, P. Doina, Metrics for finite Markov decision processes, in: Proceedings of the Twentieth conference on Uncertainty in artificial intelligence, AUAI Press, 2004, pp. 162–169, doi:10.1109/gamenets.2009.5137416.
[20] B. Piot, M. Geist, O. Pietquin, Bridging the gap between imitation learning and inverse reinforcement learning, IEEE Trans. Neaural Netw. Learn. Syst. 28 (8) (2016) 1814–1826.
[21] Y. Pu, Z. Gan, H. Ricardo, X. Yuan, C. Li, S. Andrew, L. Carin, Variational autoencoder for deep learning of images, labels and captions, in: Proceedings of the Advances in Neural Information Processing Systems, 2016, pp. 2352–2360.
[22] M. Riedmiller, Neural fitted q iteration first experiences with a data efficient neural reinforcement learning method, in: Proceedings of the European Conference on Machine Learning, 2005, pp. 317–328, doi:10.1007/11564096_32.
[23] L. Sergey, P. Zoran, K. Vladlen, Feature construction for inverse reinforcement learning, in: Proceedings of the Advances in Neural Information Processing Systems, 2010, pp. 1342–1350.
[24] H. Shi, L. Shi, M. Xu, K.S. Hwang, End-to-end navigation strategy with deep reinforcement learning for mobile robots, IEEE Trans. Ind. Inform. 16 (4) (2020) 2393–2402.
[25] M. Wulfmeier, P. Ondrúška, I. Posner, Maximum entropy deep inverse reinforcement learning, 2015, arXiv:1507.04888.
[26] J. Xu, L. Yao, L. Li, M. Ji, G. Tang, Argumentation based reinforcement learning for meta-knowledge extraction, Inf. Sci. 506 (2020) 258–272.
[27] J. Yu, M. Tan, H. Zhang, D. Tao, Y. Rui, Hierarchical deep click feature prediction for fine-grained image recognition, IEEE Trans. Pattern Anal. Mach. Intell. (2019).
[28] J. Yu, X. Yang, F. Gao, D. Tao, Deep multimodal distance metric learning using click constraints for image ranking, IEEE Trans. Cybern. 47 (12) (2016) 4014–4024.
[29] J. Zhang, J. Yu, D. Tao, Local deep-feature alignment for unsupervised dimension reduction, IEEE Trans. Image Process. 27 (5) (2018) 2420–2432.
[30] B.D. Ziebart, A. Maas, J. Bagnell, A.K. Dey, Maximum entropy inverse reinforcement learning, in: Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, 2008, pp. 1433–1438.