Neural Process Lett (2018) 47:891–905
https://doi.org/10.1007/s11063-017-9702-7

Model-Free Deep Inverse Reinforcement Learning by Logistic Regression

Eiji Uchibe[1,2]

Published online: 8 September 2017

Abstract This paper proposes model-free deep inverse reinforcement learning to find nonlinear reward function structures. We formulate inverse reinforcement learning as a problem of density ratio estimation, and show that the log of the ratio between an optimal state transition and a baseline one is given by a part of reward and the difference of the value functions under the framework of linearly solvable Markov decision processes. The logarithm of density ratio is efficiently calculated by binomial logistic regression, of which the classifier is constructed by the reward and state value function. The classifier tries to discriminate between samples drawn from the optimal state transition probability and those from the baseline one. Then, the estimated state value function is used to initialize the part of the deep neural networks for forward reinforcement learning. The proposed deep forward and inverse reinforcement learning is applied into two benchmark games: Atari 2600 and Reversi. Simulation results show that our method reaches the best performance substantially faster than the standard combination of forward and inverse reinforcement learning as well as behavior cloning.

Keywords Inverse reinforcement learning · Deep learning · Density ratio estimation · Logistic regression

This paper is based on results obtained from a project commissioned by the New Energy and Industrial Technology Development Organization (NEDO) and JSPS KAKENHI Grant Number 17H06042. This is an extended version of our paper published in ICONIP 2016 [22].

✉ Eiji Uchibe
uchibe@atr.jp

1 Department of Brain Robot Interface, ATR Computational Neuroscience Labs., 2-2-2 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-0288, Japan

2 Neural Computation Unit, Okinawa Institute of Science and Technology Graduate University, 1919-1 Tancha, Onna-son, Okinawa 904-0495, Japan

1 Introduction

Reinforcement learning (RL) provides a computational framework to find an optimal policy that achieves given goals using a trial-and-error process. Recently, RL algorithms are combined with deep learning framework and obtains human-level control policies in some Atari 2600 games [13] and Go [18]. However, it is still difficult to find an optimal policy if a reward signal from an environment is too sparse and delayed. In practice, the initial policy was trained by the form of supervised learning from expert human moves in the case of Go. Inverse reinforcement learning (IRL), which is a method of estimating a reward function that can explain a given agent's behavior [15,33], provides a computational scheme to implement imitation learning. It is also a promising approach for understanding the learning processes of biological systems such as driving a vehicle [10,17] and playing table tennis [14] because the reward specifies the goal of the behavior. If IRL finds a dense reward function from a set of optimal behaviors provided by an expert, it improves the learning process significantly to find an optimal policy.

Previously, we developed IRL under the Linearly solvable Markov Decision Process (LMDP) [21] that directly estimates the state-dependent reward and the state value function [23] simultaneously. This method exploits the fact that the logarithm of the ratio between an optimal and a baseline state transition is represented by a state-dependent reward and the difference of the value functions under the LMDP framework and they are efficiently estimated by logistic regression to classify whether the data are sampled from the optimal transition probability. Unlike most previous IRL methods such as Maximum Entropy-based IRL (MaxEnt-IRL) [33], our IRL does not need to find an optimal policy for every iteration. However, most previous studies (including our method) use linear function approximators in which a set of basis functions are prepared manually.

This paper extends our previous method [23] by introducing deep learning frameworks to identify the nonlinear representation of reward and value functions. The application of deep learning frameworks is straightforward, and the network structure of binary classifiers is derived from the simplified Bellman equation under LMDP. We also show that our IRL method is well integrated with the dueling network architecture proposed by Wang et al. [27], which explicitly separates the representation of state value and action advantage to compute a state-action value function. Our proposed IRL method initializes the part of the dueling network from the datasets from experts and it significantly improves the learning performance.

We evaluate the combination of our IRL method and the dueling network on the Atari 2600 games provided by the Arcade Learning Environment [2] and compare our method with Path Integral with LOCal optimality (PI_LOC) that is a sampling-based MaxEnt-IRL [9]. Simulation results show that the proposed method achieves better scores than simple behavior cloning. We also show that the learning speed is significantly accelerated when the estimated state value function is used with the original reward of the games. Next, the proposed method is compared with the model-based Wulfmeier's method on the game of Reversi [24,25]. We show that the performance achieved by the proposed method resembles that of model-based method with less computing time. In addition, the policy trained with the estimated reward and state value function outperforms policies that are used to collect data.

1.1 Related Work

Recently, Wulfmeier et al. [29–31] proposed deep IRL, which combined MaxEnt-IRL with a deep neural network architecture to find nonlinear reward functions. However, their method suffers from the same three problems as MaxEnt-IRL. First, deep MaxEnt-IRL is basically a model-based approach, and therefore, an environmental state transition probability is assumed to be known in advance. Second, MaxEnt-IRL requires a routine to solve forward RL problems with estimated reward functions for every iteration step. This process is usually very time-consuming and computationally expensive, even when a model of the environment is available. Finally, MaxEnt-IRL is data inefficient because it requires a set of trajectories to estimate reward. In general, a dataset consists of different length trajectories before termination, and the estimation process is sensitive to the variation.

To overcome the first and second problems of MaxEnt-IRL, relative entropy-based inverse reinforcement learning (RelEnt-IRL) [4] and Path Integral with LOCal optimality (PI_LOC) [9] introduce a sampling distribution to handle unknown environmental state transition probability. More specifically, RelEnt-IRL uses a uniform sampling distribution while PI_LOC uses the distribution that lies in the vicinity of the demonstration in order to evaluate the partition function of the probability distribution of the demonstration. Finn et al. proposed Guided Cost Learning (GCL) that improves sampling efficiency of deep MaxEnt-IRL [5]. As opposed to RelEnt-IRL and PI_LOC, GCL adapts the sampling distribution using policy optimization. They are model-free MaxEnt-IRL, but they would still suffer from the third problem.

Generative Adversarial Imitation Learning (GAIL) [8] and its model-based version [1] extract a policy directly from data as if it were obtained by reinforcement learning following inverse reinforcement learning. Their formulation is similar to Generative Adversarial Network [6] and the discriminator of GAIL distinguishes between unnormalized distribution of state-action pairs of an optimal and a learned policy. Therefore, GAIL is similar to our method because our method discriminates between data from the optimal and baseline state transition. A potential difficulty is that GAIL should estimate the normalized distribution defined over the state and action space. On the other hand, our method maintains functions defined over the state space, which are usually easier to approximate.

2 Linearly Solvable Markov Decision Process

Let $\mathcal{X}$ and $\mathcal{A}$ respectively be continuous state and discrete action spaces. At time step t, a learning agent observes environmental current state $\mathbf{x}_t \in \mathcal{X}$ and executes action $a_t \in \mathcal{A}$ that is sampled according to a stochastic policy $\pi(a_t \mid \mathbf{x}_t)$. Consequently, an immediate reward $r(\mathbf{x}_t, a_t)$ is given by the environment and the environment makes a state transition based on state transition probability $P_T(\mathbf{y}_t \mid \mathbf{x}_t, a_t)$ from $\mathbf{x}_t$ to $\mathbf{y}_t = \mathbf{x}_{t+1} \in \mathcal{X}$ by executing the action a_t. The goal of (forward) reinforcement learning is to construct an optimal policy $\pi(a \mid \mathbf{x})$ that maximizes the given objective function. Several objective functions exist, and the most widely used one is a discounted sum of rewards given by

$$V(\mathbf{x}) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(\mathbf{x}_t, a_t) \,\middle|\, \mathbf{x}_0 = \mathbf{x}\right],$$

where $\gamma \in [0, 1)$ is called the discount factor. In the same way, the state-action value function is also defined by

$$Q(\mathbf{x}, a) = r(\mathbf{x}, a) + \gamma \mathbb{E}_{\mathbf{y} \sim P_T(\cdot \mid \mathbf{x}, a)}\left[V(\mathbf{y})\right],$$

and the advantage function is given by

$$A(\mathbf{x}, a) = Q(\mathbf{x}, a) - V(\mathbf{x}). \tag{1}$$

The optimal state value function for the discounted reward setting satisfies the following Bellman equation:

$$V(\mathbf{x}) = \max_a \left[r(\mathbf{x}, a) + \gamma \mathbb{E}_{\mathbf{y} \sim P_T(\cdot|\mathbf{x},a)} \left[V(\mathbf{y}) \right] \right]. \tag{2}$$

Equation (2) is a nonlinear equation due to the max operator.

The Linearly solvable Markov Decision Process (LMDP), also known as KL-control, simplifies Eq. (2) under some assumptions [21]. LMDP's key trick is to directly optimize the state transition probability instead of the policy. More specifically, two conditional probability density functions are introduced. One is the controlled probability denoted by $\pi(\mathbf{y} \mid \mathbf{x})$, which can be interpreted as an optimal state transition. The other is the uncontrolled probability denoted by $b(\mathbf{y} \mid \mathbf{x})$, which can be regarded as an innate state transition of the target system. Theoretically, $b(\mathbf{y} \mid \mathbf{x})$ is arbitrary and can be constructed by

$$b(\mathbf{y} \mid \mathbf{x}) = \sum_a P_T(\mathbf{y} \mid \mathbf{x}, a) b(a \mid \mathbf{x}),$$

where $b(a \mid \mathbf{x})$ is a uniformly random policy.

Then the reward function is restricted to the following form:

$$r(\mathbf{x}, a) = q(\mathbf{x}) - \frac{1}{\beta} \mathrm{KL}(\pi(\cdot \mid \mathbf{x}) \parallel b(\cdot \mid \mathbf{x})), \tag{3}$$

where $q(\mathbf{x})$, β, and $\mathrm{KL}(\pi(\cdot \mid \mathbf{x}) \parallel b(\cdot \mid \mathbf{x}))$ respectively denote a state-dependent reward function, a positive inverse temperature, and the Kullback Leibler (KL) divergence between the controlled and uncontrolled state transition densities.In this case, the Bellman equation (2) is written as

$$V(\mathbf{x}) = q(\mathbf{x}) + \max_\pi \int \pi(\mathbf{y} \mid \mathbf{x}) \left[-\frac{1}{\beta} \ln \frac{\pi(\mathbf{y} \mid \mathbf{x})}{b(\mathbf{y} \mid \mathbf{x})} + \gamma V(\mathbf{y}) \right] d\mathbf{y}.$$

We can maximize the right hand side of the above equation by applying the Lagrangian method [21] to obtain the following solution:

$$\exp(\beta V(\mathbf{x})) = \exp(\beta q(\mathbf{x})) \int b(\mathbf{y} \mid \mathbf{x}) \exp(\beta\gamma V(\mathbf{y})) d\mathbf{y}. \tag{4}$$

The optimal controlled probability for the discounted reward setting is given by

$$\pi(\mathbf{y} \mid \mathbf{x}) = \frac{b(\mathbf{y} \mid \mathbf{x}) \exp(\beta\gamma V(\mathbf{y}))}{\int b(\mathbf{y}' \mid \mathbf{x}) \exp(\beta\gamma V(\mathbf{y}')) d\mathbf{y}'}. \tag{5}$$

Note that Eq. (4) remains nonlinear even though desirability function $Z(\mathbf{x}) = \exp(\beta V(\mathbf{x}))$ is introduced because of the existence of discount factor γ. In the forward reinforcement learning under the framework of LMDP, $V(\mathbf{x})$ is computed by solving Eq. (4), then $\pi(\mathbf{y} \mid \mathbf{x})$ is derived from the computed state value function.

3 Deep Inverse Reinforcement Learning

3.1 Bellman Equation for IRL

We here show a derivation of the simplified Bellman equation for inverse reinforcement learning. From Eqs. (4) and (5), we obtain the following relation for the discounted reward setting:

$$\pi(\mathbf{y} \mid \mathbf{x}) = \frac{b(\mathbf{y} \mid \mathbf{x}) \exp(\beta\gamma V(\mathbf{y}))}{\exp(\beta(V(\mathbf{x}) - q(\mathbf{x}))}.$$

The above equation shows that the optimal state transition is represented by the baseline state transition, state-dependent reward, and state value function. Taking the logarithm of both sides yields the following critical relation for the discounted reward setting [22]:

$$\frac{1}{\beta} \ln \frac{\pi(\mathbf{y} \mid \mathbf{x})}{b(\mathbf{y} \mid \mathbf{x})} = q(\mathbf{x}) + \gamma V(\mathbf{y}) - V(\mathbf{x}). \tag{6}$$

Equation (6) plays an important role in our IRL algorithms. Similar equations can be derived for average-reward, first-exit, and finite-horizon problems. Note that the right hand side of Eq. (6) is not a temporal difference error because $q(\mathbf{x})$ is the state-dependent part of the reward function in Eq. (3).

Applying the Bayes rule into Eq. (6) yields the following:

$$\ln \frac{\pi(\mathbf{x}, \mathbf{y})}{b(\mathbf{x}, \mathbf{y})} = \ln \frac{\pi(\mathbf{x})}{b(\mathbf{x})} + \beta q(\mathbf{x}) + \gamma\beta V(\mathbf{y}) - \beta V(\mathbf{x}). \tag{7}$$

Our goal is to estimate $\beta q(\mathbf{x})$ and $\beta V(\mathbf{x})$ from the observed data, and we assume two datasets of state transitions. One is $\mathcal{D}^\pi$ from the controlled probability:

$$\mathcal{D}^\pi = \left\{\left(\mathbf{x}_i^\pi, \mathbf{y}_i^\pi\right)\right\}_{i=1}^{N^\pi}, \quad \mathbf{y}_i^\pi \sim \pi(\cdot \mid \mathbf{x}_i^\pi),$$

where N^π denotes the number of data points. This is called the expert dataset. In the standard IRL setting, $\mathcal{D}^\pi$ is interpreted as data from experts to be investigated. The other is a dataset from the uncontrolled probability:

$$\mathcal{D}^b = \left\{\left(\mathbf{x}_j^b, \mathbf{y}_j^b\right)\right\}_{j=1}^{N^b}, \quad \mathbf{y}_j^b \sim b(\cdot \mid \mathbf{x}_j^b),$$

where N^b denotes the number of data points. This is called the baseline dataset. We are interested in estimating ratios $\pi(\mathbf{x})/b(\mathbf{x})$ and $\pi(\mathbf{x}, \mathbf{y})/b(\mathbf{x}, \mathbf{y})$ from $\mathcal{D}^b$ and $\mathcal{D}^\pi$. Note that our method solves two density ratio estimation problems while the standard IRL method such as MaxEnt-IRL solve one density estimation problem. In many cases, density ratio estimation is easier than density estimation [19].

3.2 LogReg-IRL: Logistic Regression-Based IRL

This subsection shows how Eq. (7) is used to estimate $\beta q(\mathbf{x})$ and $\beta V(\mathbf{x})$. LogReg, which is a density estimation method using logistic regression [3,19], is appropriate to estimate the log ratio of the following two densities: $\ln \pi(\mathbf{x})/b(\mathbf{x})$ and $\ln \pi(\mathbf{x}, \mathbf{y})/b(\mathbf{x}, \mathbf{y})$. First, to estimate $\ln \pi(\mathbf{x})/b(\mathbf{x})$, assign a selector variable $\eta = -1$ to the samples from the uncontrolled probability and $\eta = 1$ to the samples from the controlled probability:

$$b(\mathbf{x}) = \Pr(\mathbf{x} \mid \eta = -1), \quad \pi(\mathbf{x}) = \Pr(\mathbf{x} \mid \eta = 1).$$

The density ratio can be represented by applying the Bayes rule:

$$\begin{aligned}\frac{\pi(\mathbf{x})}{b(\mathbf{x})} &= \left(\frac{\Pr(\eta = 1 \mid \mathbf{x}) \Pr(\mathbf{x})}{\Pr(\eta = 1)}\right) \left(\frac{\Pr(\eta = -1 \mid \mathbf{x}) \Pr(\mathbf{x})}{\Pr(\eta = -1)}\right)^{-1} \\ &= \frac{\Pr(\eta = -1)}{\Pr(\eta = 1)} \frac{\Pr(\eta = 1 \mid \mathbf{x})}{\Pr(\eta = -1 \mid \mathbf{x})}.\end{aligned}$$

The first ratio, $\Pr(\eta = -1)/\Pr(\eta = 1)$, is estimated by N^b/N^π, and the second ratio is computed after estimating the conditional probability $\Pr(\eta \mid \mathbf{x})$ by a logistic regression classifier:

$$\Pr(\eta \mid \mathbf{x}) = \sigma\left(\eta f_x(\mathbf{x}; \mathbf{w}_x)\right),$$

where $\sigma(x) = 1/(1+\exp(-x))$ is a sigmoid function and $f_x(\mathbf{x}; \mathbf{w}_x)$ denotes a deep neural network function parameterized by the weight vector $\mathbf{w}_x$. Note that the logarithm of the density ratio is given by

$$\ln \frac{\pi(\mathbf{x})}{b(\mathbf{x})} = f_x(\mathbf{x}; \mathbf{w}_x) + \ln \frac{N^b}{N^\pi}. \tag{8}$$

Network weights $\mathbf{w}_x$ can be estimated by the backpropagation whose objective function is given by the following negative regularized log-likelihood.

$$J_x(\mathbf{w}_x) = -\frac{1}{N^b}\sum_{j=1}^{N^b} \ln \sigma\left(-f_x(\mathbf{x}_j^b; \mathbf{w}_x)\right) - \frac{1}{N^\pi}\sum_{i=1}^{N^\pi} \ln \sigma\left(f_x\left(\mathbf{x}_i^\pi; \mathbf{w}_x\right)\right) + \frac{\lambda_x}{2}\|\mathbf{w}_x\|_2^2,$$

where λ_x is a regularization constant. The closed-form solution is not derived, but it is possible to minimize it efficiently by standard nonlinear optimization methods such as backpropagation. Note that the first density ratio estimation is not necessary if we can assure $\pi(\mathbf{x}) = b(\mathbf{x})$. In this case, we do not need to evaluate the density ratio network because $\ln \pi(\mathbf{x})/b(\mathbf{x}) = 0$.

Next, $\ln \pi(\mathbf{x}, \mathbf{y})/b(\mathbf{x}, \mathbf{y})$ is estimated by Eq. (7) in the same way. Nonlinear function approximators for $\beta q(\mathbf{x})$ and $\beta V(\mathbf{x})$ are respectively introduced by

$$\beta q(\mathbf{x}) \approx r_{\text{est}}(\mathbf{x}; \mathbf{w}_r), \quad \beta V(\mathbf{x}) \approx V_{\text{est}}(\mathbf{x}; \mathbf{w}_V), \tag{9}$$

where $r_{\text{est}}(\mathbf{x}, \mathbf{w}_r)$ and $V_{\text{est}}(\mathbf{x}, \mathbf{w}_V)$ denote the deep neural network of the state dependent reward parameterized by the weights $\mathbf{w}_r$ and that of the state value function parameterized by $\mathbf{w}_V$, respectively. By substituting Eqs. (8) and (9) into Eq. (7), we obtain the following relationship:

$$\ln \frac{\pi(\mathbf{x}, \mathbf{y})}{b(\mathbf{x}, \mathbf{y})} = f_x(\mathbf{x}; \mathbf{w}_x) + r_{\text{est}}(\mathbf{x}; \mathbf{w}_r) + \gamma V_{\text{est}}(\mathbf{y}; \mathbf{w}_V) - V_{\text{est}}(\mathbf{x}; \mathbf{w}_V) + \ln \frac{N^b}{N^\pi}.$$

The above equation is also interpreted as a density ratio estimation problem, and network parameters $\mathbf{w}_r$ and $\mathbf{w}_V$ are estimated by logistic regression in which the classifier is given by

$$\Pr(\eta \mid \mathbf{x}, \mathbf{y}) = \sigma\left(\eta\left(f_x(\mathbf{x}; \mathbf{w}_x) + r_{\text{est}}(\mathbf{x}; \mathbf{w}_r) + \gamma V_{\text{est}}(\mathbf{y}; \mathbf{w}_V) - V_{\text{est}}(\mathbf{x}; \mathbf{w}_V)\right)\right). \tag{10}$$

Note that the inverse temperature β is not estimated as an independent parameter. The parameter vectors, $\mathbf{w}_r$ and $\mathbf{w}_V$, are optimized by standard logistic regression algorithms in the same way as $\mathbf{w}_x$.

Figure 1 shows our proposed deep neural network for inverse reinforcement learning. The input consists of current state $\mathbf{x}$ and next state $\mathbf{y}$, and the output consists of label η that addresses whether $(\mathbf{x}, \mathbf{y})$ are given from $\mathcal{D}^\pi$ or $\mathcal{D}^b$. The network has three sub-networks: density-ratio network $f_x(\mathbf{x}; \mathbf{w}_x)$, reward network $r_{\text{est}}(\mathbf{x}; \mathbf{w}_r)$, and state value function network $V_{\text{est}}(\mathbf{x}; \mathbf{w}_V)$. Note that state $\mathbf{y}$ is given only to the value network. After the sub-networks compute the log of the density ratio, the reward, and the value functions, the log of the density ratio is computed by the Bellman equation (7) and used to compute the classifier's probability (10).

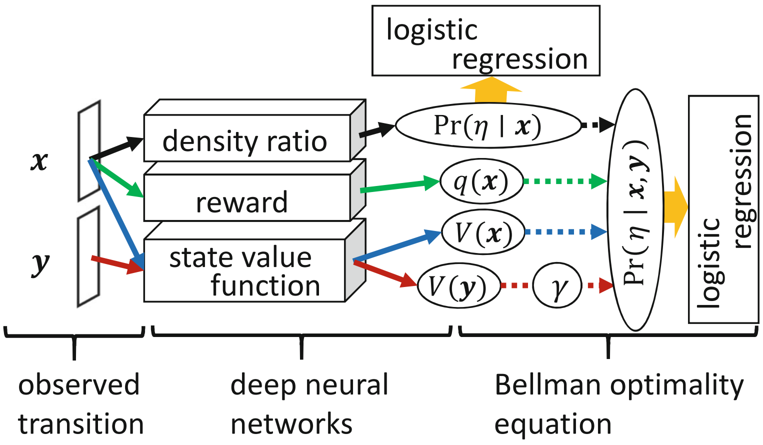

Fig. 1 Proposed network architecture for inverse reinforcement learning that consists of three sub-networks: density ratio, reward, and state value function. Then the Bellman equation (7) is computed from the outputs of the three networks

3.3 Forward Reinforcement Learning

This section explains how the result of inverse reinforcement learning is used to find an optimal policy by forward reinforcement learning. Because our IRL method estimates the state value function, it should be used to initialize a part of the deep neural networks for forward reinforcement learning. We adopt the dueling network that represents state values and state-dependent action advantages separately [27]. Using the definition of the advantage function (1), the state-action value function is decomposed into two functions:

$$Q(\mathbf{x}, a) = V(\mathbf{x}) + A(\mathbf{x}, a). \tag{11}$$

However, we cannot utilize the decomposition because V and A are not recovered from Q uniquely. Therefore, Wang et al. [27] propose to impose a constraint on the advantage function. Since the advantage function measures whether or not the action is better or worse than the policy's average action, it is represented by

$$A(\mathbf{x}, a) = P(\mathbf{x}, a) - \frac{1}{|\mathcal{A}|} \sum_{a'} P(\mathbf{x}, a'), \tag{12}$$

where $P(\mathbf{x}, a)$ is the preference function and $|\mathcal{A}|$ denotes the number of actions. Note that the second term in the right hand side of Eq. (12) is the average preference value under a uniformly random policy. As noted by Wang et al. [27], the advantage function computed by Eq. (12) is slightly different from Eq. (1), but it is reported that using Eq. (12) increases the stability of the optimization. Figure 2 shows the deep neural network for forward reinforcement learning. The input is the current state $\mathbf{x}$ and the output consists of the state action value function. The network has three sub-networks: reward network $r_{\text{est}}(\mathbf{x}; \mathbf{w}_r)$, state value function network $V_{\text{est}}(\mathbf{x}; \mathbf{w}_V)$ and the preference network. To optimize the state value and preference function networks, many algorithms such as SARSA, Deep Q Network (DQN) [13], Double DQN [26] can be used and we adopt DQN for simplicity.

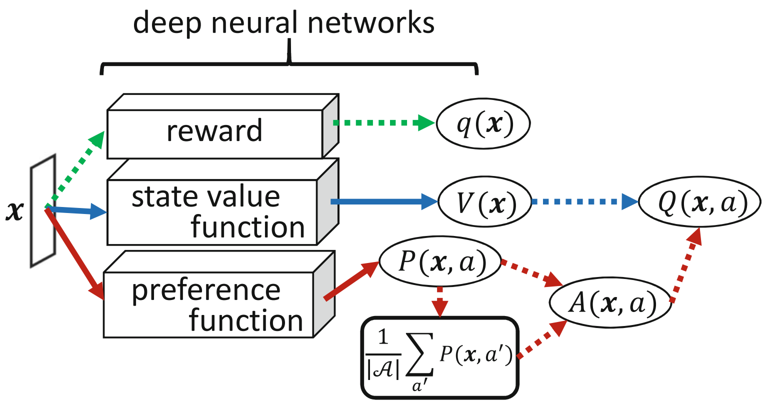

Fig. 2 Proposed network architecture for forward reinforcement learning that consists of two sub-networks: state value function and preference. The advantage function is derived from the preference function by Eq. (12). The output of the network is the state-action value function given by Eq. (11)

4 Experiments

4.1 Atari Games

To validate our method, we selected Atari 2600 games provided by the Arcade Learning Environment [2], which has been a popular benchmark for evaluating reinforcement learning progress in recent years [7,13,27]. In particular, we chose six games: Boxing, Breakout, Enduro, Pong, Seaquest, and Space Invaders.

To collect the expert dataset, we generated 1.5 million frames of human gameplay experiences in the format of

$$\mathcal{T} = \{\tau_i\}_{i=1}^{\tilde{N}^\pi}, \quad \tau_i = \left\{\left(\mathbf{x}_{i,t}^\pi, a_{i,t}^{\text{human}}, \mathbf{x}_{i,t+1}^\pi\right)\right\}_{t=0}^{T_i-1},$$

where $a_{i,t}^{\text{human}}$ is an action executed by a subject. We collected the data from three subjects. Then, we selected the state transition data $\{(\mathbf{x}_{i,t}^\pi, \mathbf{x}_{i,t+1}^\pi)\}$ to construct $\mathcal{D}^\pi$. On the other hand, we found that a policy that selects actions uniformly at random did not explore the environment efficiently because the game was over at the early stage of game plays. Therefore, for every state $\mathbf{x}_t$ in $\mathcal{D}^\pi$ a uniformly random action a_t^{random} is executed, and sample the next state. In other words, the baseline dataset is constructed by

$$\mathcal{D}^b = \left\{\left(\mathbf{x}_{i,}, \tilde{\mathbf{x}}_{i,t+1}\right)\right\}, \quad \tilde{\mathbf{x}}_{i,t+1} \sim P_T\left(\cdot \mid \mathbf{x}_{i,t}, a_{i,t}^{\text{random}}\right).$$

In this sampling method, we assured $\pi(\mathbf{x}) = b(\mathbf{x})$ and we removed the density ratio network from the entire networks shown in Fig. 1. Although this sampling method is not always possible for real-world applications, a similar method is used in PI_LOC [9] that is interpreted as a sampling-based MaxEnt-IRL.

We follow the basic setup of previous studies [7,13,26,27] to implement the deep neural networks of inverse reinforcement learning. In each sub-network, there are three convolutional layers followed by two fully-connected layers. The first convolutional layer has 32 8×8 filters with stride 4, the second 64 4×4 filters with stride 2, and the third and final convolutional layer has 64 3×3 filters with stride 1. The deep neural networks of forward reinforcement learning has the same low-level convolutional structure of that of inverse reinforcement learning. The preference function network has a fully connected layer with 512

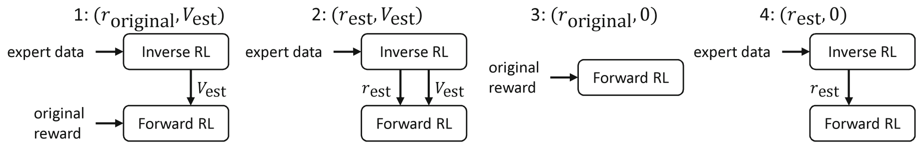

Fig. 3 Four forward reinforcement learning settings. Expert data is converted to r_{est} and V_{est} by the proposed IRL and they are utilized by the settings (r_{original}, V_{est}), (r_{est}, V_{est}), and (r_{est}, 0) while the setting (r_{original}, 0) is the standard forward reinforcement learning scenario.

units. The final hidden layers of the preference function are both fully-connected with $|\mathcal{A}|$ neurons.

Since IRL is an ill-posed problem and the reward function is not uniquely determined, we cannot compare the estimated reward function with the original reward function of the game. However, we could not compare our method with deep MaxEnt-IRL [29–31] because deep MaxEnt-IRL has to find an optimal policy for every iteration and it took enormous time on the games of Atari 2600. Therefore, we selected PI_LOC [9] for comparison because $\mathcal{D}^b$ can be used to evaluate the partition function. Instead of solving forward reinforcement learning, PI_LOC evaluates the partition function from a set of trajectories generated by the baseline state transition given by $\mathcal{T}^b = \{\tau_j\}_{j=1}^{\tilde{N}^b}$, where $\tau_j = \mathbf{x}_{j,0}^b, \mathbf{x}_{j,1}^b, \ldots, \mathbf{x}_{j,T_j}^b$. The log likelihood is formulated by

$$L(\tau; \mathbf{w}_r) = \sum_{i=1}^{\tilde{N}^\pi} \ln \frac{\exp\left(\sum_t \gamma^t r_{\text{est}}\left(\mathbf{x}_{i,t}^\pi; \mathbf{w}_r\right)\right)}{\sum_{j=1}^{\tilde{N}^b} \exp\left(\sum_t \gamma^t r_{\text{est}}\left(\mathbf{x}_{j,t}^b; \mathbf{w}_r\right)\right)}.$$

We used $\mathcal{D}^b$ to construct $\mathcal{T}^b$. This is reasonable because trajectories in $\mathcal{T}^b$ should be noisy trajectories around the optimal state transitions. Note that $|\mathcal{T}^\pi|$ is much smaller than $|\mathcal{D}^\pi|$ because an element in $\mathcal{T}^\pi$ is a trajectory. To evaluate the quality of the estimated reward and state value function, we computed an optimal policy by forward reinforcement learning with them and compared the learning speed. We considered four learning settings illustrated in Fig. 3:

1. (r_{original}, V_{est}): the value function network is initialized by the result of IRL and the original reward is used to find a policy by forward reinforcement learning. That is, $\mathbf{w}_V$ is initialized by the weight vector estimated by IRL and the reward network is not used at all.
2. (r_{est}, V_{est}): the value function network is initialized by the result of IRL and the estimated reward is used.
3. (r_{original}, 0): the value function is initialized by zero ($\mathbf{w}_V = \mathbf{0}$) and the original reward is used. This corresponds to a standard forward reinforcement learning scenario.
4. (r_{est}, 0): the value function is initialized by zero and the estimated reward is used. Only this setting is feasible in the standard IRL methods, because the state value function is not estimated explicitly.
5. ($r_{\text{PI\_LOC}}$, 0): the reward is estimated by PI_LOC and the value function is initialized by zero.
6. CLONING: a stochastic policy $\pi(a \mid \mathbf{x})$ is directly estimated by maximizing the likelihood from the optimal dataset.

To train the deep neural networks of forward reinforcement learning, we use the identical learning algorithm, experience replay, clipping the gradient used by Wang et al. [27]. The

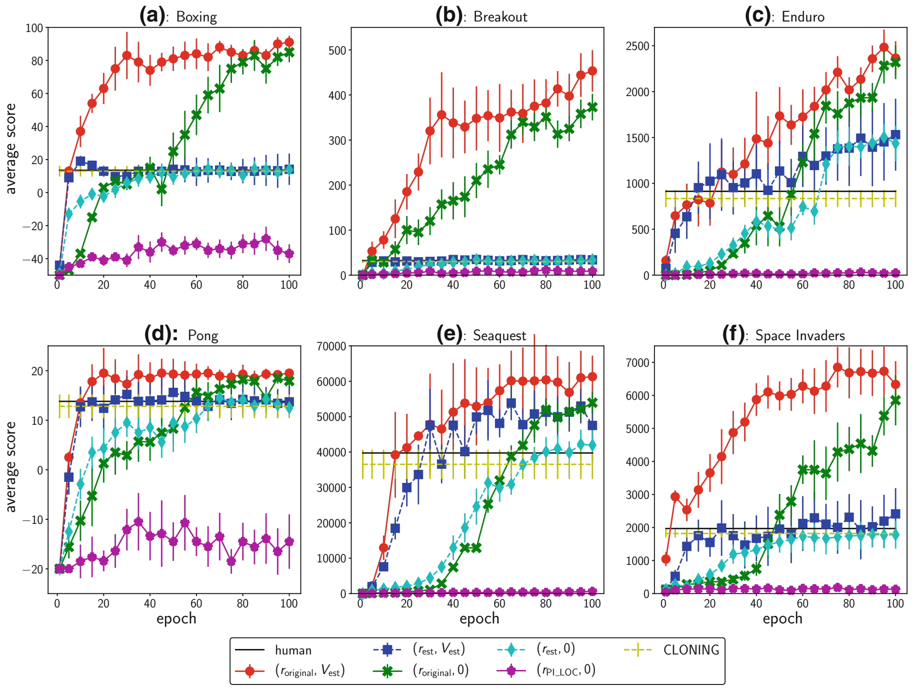

Fig. 4 Average performance of forward reinforcement learning with the different settings over ten experiments when tested on a set of six Atari games. Vertical bars represent one standard deviation from the mean

major difference is an action selection strategy and we adopted softmax action selection in which a stochastic policy is calculated by $\pi(a \mid \mathbf{x}) \propto \exp(Q(\mathbf{x}, a)/T)$, where T is a temperature that controls the randomness. According to the previous studies [26,27], we start the game with up to 30 no-op actions to provide random starting positions for the agent. The discount factor is set to 0.99.

Figure 4 compares the learning speed, in which the performance is evaluated by the score of the game. A training epoch is 250,000 frames and for every 10 epochs we evaluate the network of forward reinforcement learning with a testing epoch that lasts 125,000 frames. The horizontal solid and dotted lines represent the performance of human experts and CLONING, respectively. The error bars in each panel shows one standard deviation from the mean scores of CLONING and those of forward reinforcement learning methods, respectively.

In all of the six game environments, the best learning performance was achieved by the setting $(r_{\text{original}}, V_{\text{est}})$. That is, we found the best performance when the state value function of forward reinforcement learning was initialized by the result of inverse reinforcement learning and the original game reward was used. As compared with the standard forward reinforcement learning setting $(r_{\text{original}}, 0)$, the learning speed was significantly improved. On the other hand, the policies trained with the estimated reward r_{est} was poorer than those trained with the original reward r_{original} at the final performance at the end of training. This is not surprising, because the expert datasets provided by the subjects were not optimal. However, we also found that the setting $(r_{\text{est}}, V_{\text{est}})$ achieved slightly better scores than the human plays on the games of Enduro and Seaquest. This suggests that our proposed method retrieved more useful information from the dataset. In addition, the learning speed was improved by initializing the state value function when we compared the speed of $(r_{\text{est}}, V_{\text{est}})$ with that of $(r_{\text{est}}, 0)$. The performance of PI_LOC was extremely poor in this experiment. One possible reason is that PI_LOC is sensitive to the trajectory length and its variation. $\sum_t \gamma^t r_{\text{est}}(\mathbf{x}_{i,t}; \mathbf{w}_r)$ is a Monte Carlo estimate of the value of the trajectory τ_i and it has large variance for small $\tilde{N}^\pi$ and large T_i.

4.2 Reversi

Reversi (a.k.a. Othello), which is a deterministic, perfect information, zero-sum game for two players, has been studied by the AI community [11,12,20,24,25,32]. The game's goal is to control a majority of the pieces at the end of the game by forcing as many of your opponent's pieces to be turned over on an 8×8 board as possible. A single move might change up to 20 pieces, and an average of 60 moves are needed to complete the game. Although reinforcement learning has been successfully applied to the game of GO, which is much more complicated than Reversi, it took a huge amount of computing time to find an optimal policy [18]. This provides motivation for finding a dense reward structure by IRL.

To prepare $\mathcal{D}^\pi$, we used three stationary policies: the first policy is RANDOM, which always selects a random action from available actions. The other players are HEUR and COEV that take an action based on the following evaluation function

$$F(\mathbf{x}) = \sum_{i=1}^{64} x_i w_i + n, \quad n \sim U(-2, 2)$$

where x_i is 1 when the square i is occupied by the player's own disc, -1 when it is occupied by an opponent's disc and 0 when it is not occupied, and w_i is the weight. Notice that n is a random variable that is sampled from a uniform distribution $U(a, b)$ on the interval (a, b). Adding a noise to the evaluation is for the reason that we would like to collect a variety of game trajectories. The weight w_i of HEUR is determined manually while that of COEV is optimized by a co-evolutionary computation method [11]. Every policy repeatedly played against every other, and then the state transitions were retrieved from the game trajectories of the winners. On the other hand, the baseline dataset $\mathcal{D}^b$ was constructed by retrieving the state transitions from the trajectories in which two RANDOM policies played the game. The numbers of samples are $|\mathcal{D}^\pi| = |\mathcal{D}^b| = 10^5$.

In this experiment, each sub-network of IRL is implemented by a feedforward neural network with two hidden layers [24,25] with rectified linear units and one linear output layer. As opposed to the previous experiment, the density ratio network is important because we cannot assume $\pi(\mathbf{x}) = b(\mathbf{x})$. The input is given by a 64-node vector $\mathbf{x} \in \{-1, 0, +1\}^{64}$, where 0 represents an empty square, and $+1$ and -1 respectively represent the black and white pieces. Every hidden layer has 100 nodes. Since the computational cost of finding an optimal policy on the game of Reversi is much cheaper than that on the Atari 2600 games, we can evaluate the model-based Wulfmeier's method that computes the partition function with dynamic programming. Table 1 compares the approximate computing time for estimating reward between the proposed method, Wulfmeier's model-based deep MaxEnt-IRL, and PI_LOC, where the code is written in python and executed on a Ubuntu Linux 14.04 LTS AMD64 systems, using Intel Xeon E5-1620V30 CPU with 32 GB RAM and Tesla K40C. PI_LOC needed less computing time than the proposed method because PI_LOC does not need to estimate the state value function as well as the density ratio. Apparently, the computing time was much shorter than that of Wulfmeier's method because our method and PI_LOC did not solve forward reinforcement learning.

Table 1 Comparison of approximate computation time (hours) between the proposed method, Wulfmeier's model-based MaxEnt-IRL, and PI_LOC

Proposed method	4.5
Wulfmeier's method (model-based MaxEnt-IRL)	38.5
PI_LOC (sampling-based MaxEnt-IRL)	1.7

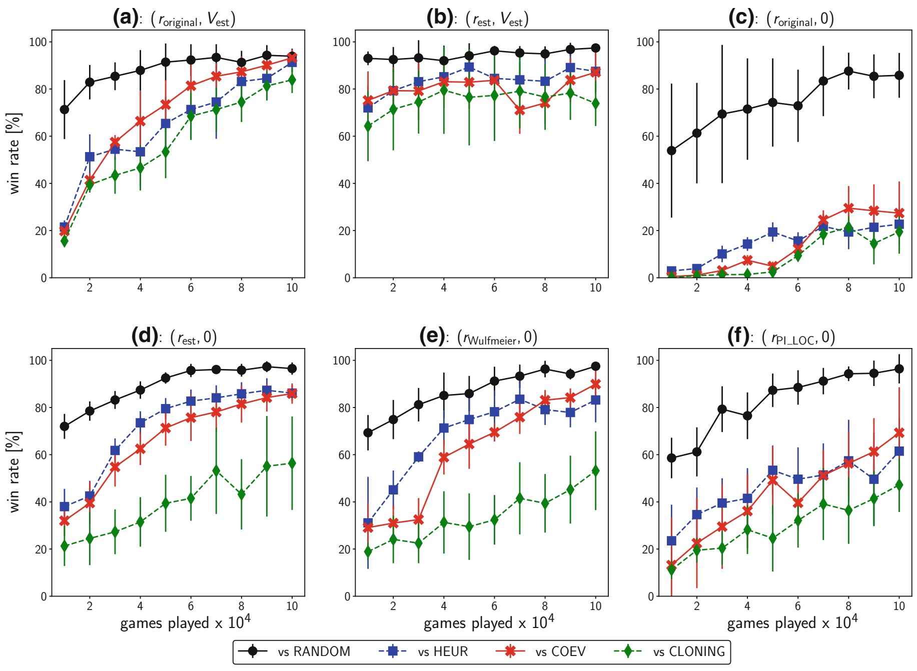

Fig. 5 Average performance of forward reinforcement learning with the different settings over ten experiments when tested versus fixed stationary policies RANDOM, HEUR, COEV and CLONING

As we did in the previous Sect. 4.1, we evaluated seven policies optimized by the following settings (1) $(r_{\text{original}}, V_{\text{est}})$, (2) $(r_{\text{est}}, V_{\text{est}})$, (3) $(r_{\text{original}}, 0)$, (4) $(r_{\text{est}}, 0)$, (5) $(r_{\text{Wulfmeier}}, 0)$,(6) $(r_{\text{PI\_LOC}}, 0)$, and (7) CLONING, where the fifth setting means that the value function is initialized by zero while the reward function estimated by Wulfmeier's method was used to train the deep networks of forward reinforcement learning. Notice that a policy obtained by CLONING is interpreted as a mixed policy of RAND, HEUR, and COEV. The original reward associated with a terminal state is 1, 0, and −1, respectively, for win, tie, and lose. The discount factor is set to 0.99.

Figure 5 illustrates how the win rate develops during training when tested versus fixed stationary policies RANDOM, HEUR, COEV, and CLONING. Interestingly, the best learning speed and the best win rate were achieved by the setting $(r_{\text{est}}, V_{\text{est}})$. The policy trained with the setting $(r_{\text{original}}, V_{\text{est}})$ outperformed all of the four policies, but it took about 3×10^4 plays to defeat HEUR and COEV and 5.5×10^4 plays to defeat CLONING. A comparison between (a) and (c) and that between (b) and (d) revealed that initializing a state value function by the proposed method was quite effective for faster learning. In fact, CLONING outperformed the policy trained with the setting $(r_{\text{original}}, r_{\text{est}})$ because a poor policy at the early stage of

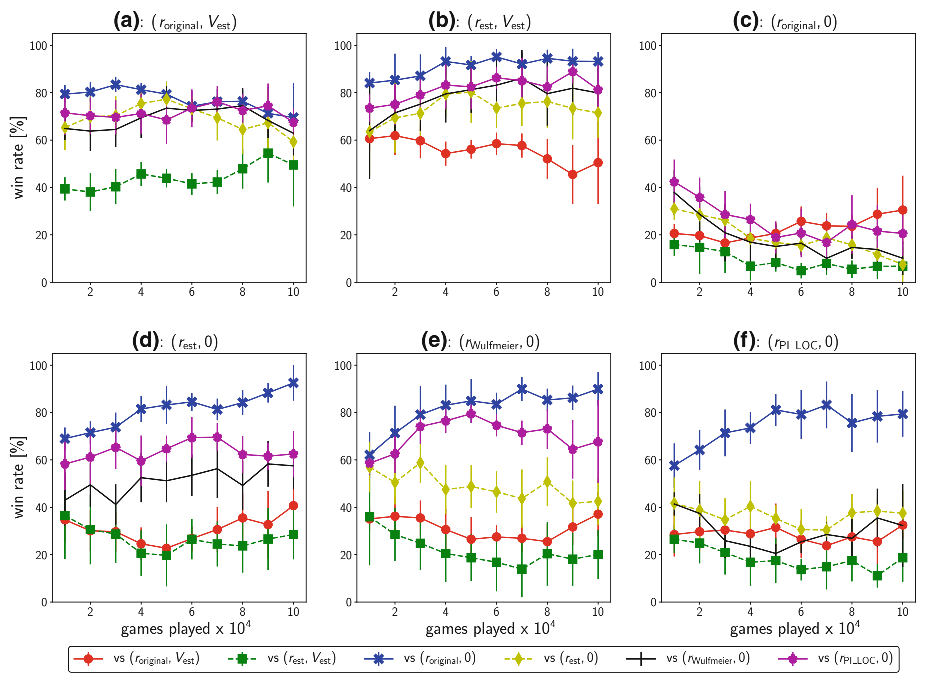

Fig. 6 Average performance of forward reinforcement learning with the different settings over ten experiments when tested versus learning agent

learning failed to collect good experiences. There were no significant difference between (d) and (e), and it suggests that the reward estimated by the proposed method was similar to that by Wulfmeier's method from a viewpoint of learned policies at the end. On the contrary, Fig. 5f shows that the performance of model-free PI_LOC was slightly worse than that of the model-based Wulfmeir's method. This is not surprising because it was relatively easy to construct the state transition dynamics of the Reversi even though the opponent was not modeled explicitly.

Figure 6 depicts how the win rate develops during training when tested versus learning agents. We did not show the result of learning from self-play because the win rate was always 100%. Note that these experiments do not satisfy the Markovian assumption because two learning agents interacted at the same time. Even if the opponent was a learning agent, the best learning speed and the best win rate were achieved by the setting (r_{est}, V_{est}). Figure 6c shows that the agent trained with the original reward could not win the agents trained with the reward estimated by the IRL methods. Since the same learning algorithm was used, the results should be explained by the difference of reward functions. One possible reason is that r_{est}, $r_{\text{Wulfmeier}}$, and $r_{\text{PI\_LOC}}$ had a dense structure that provides more information about the game. Consequently, the agents trained with those reward functions learned faster than the agent with r_{original}. There were no significant difference between (d) and (e) as we observed in Fig. 5.

5 Conclusion

We proposed model-free deep inverse reinforcement learning that can estimate the state value function as well as the reward function. Our IRL is efficiently solved by binomial logistic regression and we do not need to solve forward reinforcement learning. Estimating the state value function contributes to the fast convergence because the forward reinforcement learning process with the dueling network architecture can exploit the estimated state value function to initialize the weights of the network. The proposed method can be applied into continuous action problems without any changes because the dataset does not contain any action. In this case, we have to replace DQN with other algorithms that can handle continuous actions such as Trust Region Policy Optimization [16].

In order to improve the learning process of forward reinforcement learning, we used the shaping reward technique [15] to exploit the result of inverse reinforcement learning [22]. It computes shaping rewards by $r_{\text{est}}(\mathbf{x}) + \gamma V_{\text{est}}(\mathbf{y}) - V_{\text{est}}(\mathbf{x})$, and the standard DQN is adopted to find a policy, and we reported that this approach also improved the learning process. However, using the shaping reward is essentially equivalent to initializing the value function by V_{est} [28], and the dueling network shown in Fig. 2 directly utilized the result of inverse reinforcement learning. Furthermore, we did not optimize the network architecture in this study. More systematic investigations on such deep networks as activation functions and learning algorithms are needed for future study.

A major limitation of our method is that the estimated reward and value function severely depend on the choice of the baseline dataset. To overcome this limitation, we plan to introduce the framework of GCL and GAIL that are based on Generative Adversarial Networks. If forward reinforcement learning with an estimated reward generates a set of baseline state transitions, we can devise a new method that can learn reward and baseline state transition simultaneously based on our current framework. This extension is left for future work.

Acknowledgements We would like to thank Prof. Kenji Doya for helpful comments.

References

1. Baram N, Anschel O, Caspi I, Mannor S (2017) End-to-end differentiable adversarial imitation learning. In: Proceedings of the 34th international conference on machine learning, pp 390–399
2. Bellemare MG, Naddaf Y, Veness J, Bowling M (2013) The arcade learning environment: an evaluation platform for general agents. J Artif Intell Res 47:253–279
3. Bickel S, Brückner M, Scheffer T (2009) Discriminative learning under covariate shift. J Mach Learn Res 10:2137–2155
4. Boularias A, Kober J, Peters J (2011) Relative entropy inverse reinforcement learning. In: Proceedings of the 14th international conference on artificial intelligence and statistics
5. Finn C, Levine S, Abbeel P (2016) Guided cost learning: deep inverse optimal control via policy optimization. In: Proceedings of the 33rd international conference on machine learning, pp 49–58
6. Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y (2014) Generative adversarial nets. Adv Neural Inf Process Syst 27:2672–2680
7. Guo X, Singh S, Lewis RL, Lee H (2016) Deep learning for reward design to improve Monte Carlo tree search in ATARI games. In: Proceedings of the 25th international joint conference on artificial intelligence
8. Ho J, Ermon S (2016) Generative adversarial imitation learning. Adv Neural Inf Process Syst 29:4565–4573

9. Kalakrishnan M, Pastor P, Righetti L, Schaal S (2013) Learning objective functions for manipulation. In: Proceedings of IEEE international conference on robotics and automation, pp 1331–1336
10. Kuderer M, Gulati S, Burgard W (2015) Learning driving styles for autonomous vehicles from demonstration. In: Proceedings of IEEE international conference on robotics and automation, pp 2641–2646
11. Lucas S, Runarsson T (2006) Temporal difference learning versus co-evolution for acquiring Othello position evaluation. In: Proceedings of IEEE symposium on computational intelligence and games, pp 52–59
12. Miyazaki K, Kobayashi S (2004) Development of a reinforcement learning system to play Othello. Artif Life Robot 7:177–181
13. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumaran D, Wierstra D, Legg S, Hassabis D (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529–533
14. Muelling K, Boularias A, Mohler B, Schölkopf B, Peters J (2014) Learning strategies in table tennis using inverse reinforcement learning. Biol Cybern 108(5):603–619
15. Ng AY, Russell S (2000) Algorithms for inverse reinforcement learning. In: Proceedings of the 17th international conference on machine learning, pp 663–670
16. Schulman J, Levine S, Abbeel P, Jordan M, Moritz P (2015) Trust region policy optimization. In: Proceedings of the 32nd international conference on machine learning, pp 1889–1897
17. Shimosaka M, Nishi K, Sato J, Kataoka H (2015) Predicting driving behavior using inverse reinforcement learning with multiple reward functions towards environmental diversity. In: Proceedings of IEEE intelligent vehicles symposium, pp 567–572
18. Silver D, Huang A, Maddison CJ, Guez A, Sifre L, van den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M, Dieleman S, Grewe D, Nham J, Kalchbrenner N, Sutskever I, Lillicrap T, Leach M, Kavukcuoglu K, Graepel T, Hassabis D (2016) Mastering the game of go with deep neural networks and tree search. Nature 529(7587):484–489
19. Sugiyama M, Suzuki T, Kanamori T (2012) Density ratio estimation in machine learning. Cambridge University Press, Cambridge
20. Szubert M, Jaśkowski W, Krawiec K (2013) On scalability, generalization, and hybridization of coevolutionary learning: a case study for Othello. IEEE Trans Comput Intell AI Games 5(3):214–226
21. Todorov E (2009) Efficient computation of optimal actions. PNAS 106(28):11478–11483
22. Uchibe E (2016) Deep inverse reinforcement learning by logistic regression. In: Proceedings of the 23rd international conference on neural information processing
23. Uchibe E, Doya K (2014) Inverse reinforcement learning using dynamic policy programming. In: Proceedings of the 4th international conference on development and learning and on epigenetic robotics, pp 222–228
24. van den Dries S, Wiering MA (2012) Neural-fitted TD-leaf learning for playing Othello with structured neural networks. IEEE Trans Neural Netw Learn Syst 23(11):1701–1713
25. van der Ree M, Wiering M (2013) Reinforcement learning in the game of Othello: learning against a fixed opponent and learning from self-play. In: Proceedings of IEEE symposium on adaptive dynamic programming and reinforcement learning, pp 108–115
26. van Hasselt H, Guez A, Silver D (2016) Deep reinforcement learning with double Q-learning. In: Proceedings of the 30th AAAI conference on artificial intelligence
27. Wang Z, Schaul T, Hessel M, van Hasselt H, Lanctot M, de Freitas N (2016) Dueling network architectures for deep reinforcement learning. In: Proceedings of the 33rd international conference on machine learning
28. Wiewiora E (2003) Potential-based shaping and Q-value initialization are equivalent. J Artif Intell Res 19:205–208
29. Wulfmeier M, Ondrúška P, Posner I (2015) Maximum entropy deep inverse reinforcement learning. In: NIPS Deep Reinforcement Learning Workshop
30. Wulfmeier M, Rao D, Posner I (2016a) Incorporating human domain knowledge into large scale cost function learning. In: NIPS deep reinforcement learning workshop
31. Wulfmeier M, Wang DZ, Posner I (2016b) Watch this: scalable cost-function learning for path planning in urban environments. In: Proceedings of IEEE/RSJ international conference on intelligent robots and systems
32. Yoshioka T, Ishii S, Ito M (1999) Strategy acquisition for the game Othello based on reinforcement learning. IEICE Trans Inf Syst E82-D(12):1618–1626
33. Ziebart BD, Maas A, Bagnell JA, Dey AK (2008) Maximum entropy inverse reinforcement learning. In: Proceedings of the 23rd AAAI conference on artificial intelligence