Self-Improving Generative Adversarial Reinforcement Learning

Yang Liu
Teesside University
Middlesbrough, UK
y.liu@tees.ac.uk

Yifeng Zeng
Teesside University
Middlesbrough, UK
y.zeng@tees.ac.uk

Yingke Chen
Teesside University
Middlesbrough, UK
y.chen@tees.ac.uk

Jing Tang
Teesside University
Middlesbrough, UK
j.tang@tees.ac.uk

Yinghui Pan
Jiangxi Uni. of Finance&Economics
Nanchang, China
panyinghui@jxufe.edu.cn

ABSTRACT

The lack of data efficiency and stability is one of the main challenges in end-to-end model free reinforcement learning (RL) methods. Recent researches solve the problem resort to supervised learning methods by utilizing human expert demonstrations, e.g. imitation learning. In this paper we present a novel framework which builds a self-improving process upon a policy improvement operator, which is used as a black box such that it has multiple implementation options for various applications. An agent is trained to iteratively imitate behaviors that are generated by the operator. Hence the agent can learn by itself without domain knowledge from human. We employ generative adversarial networks (GAN) to implement the imitation module in the new framework. We evaluate the framework performance over multiple application domains and provide comparison results in support.

KEYWORDS

Reinforcement Learning; Generative Adversarial Nets; Imitation learning; Policy Iteration; Policy distillation; Deep Learning

ACM Reference Format:
Yang Liu, Yifeng Zeng, Yingke Chen, Jing Tang, and Yinghui Pan. 2019. Self-Improving Generative Adversarial Reinforcement Learning. In *Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019,* IFAAMAS, 9 pages.

1 INTRODUCTION

Reinforcement learning as a field has had major successes in the past few years [12, 15, 26, 30], particularly as techniques utilizing deep neural networks (DNN) have started to permeate the research community. The techniques like Deep Q Network (DQN) [14], trust region policy optimization (TRPO) [23], and asynchronous advantage actor-critic (A3C) [13] recently grow significant research contribution on deep reinforcement learning (DRL) [3].

However, there are some remaining challenges. The first one is the sample inefficiency problem. In the classic Atari game experiments (which have been one of the most widely used benchmark for DRL), training a policy by DQN to perform close to human level will normally cost many millions frame of the sample data. In other word, it requires hundreds hours of play experience on the relatively simple Atari tasks. In more complex control tasks like the ones in the MuJoCo simulator, normally thousands of GPU hours are required to reach an acceptable performance, and this corresponds to thousands year's natural human experience.

The second challenge is the design of the reward. Reinforcement learning process is guided by a predefined reward function or a reward signal. The delayed reward problem [27] always occurs while RL is employed as a tool to solve the sequential decision making problem. Sparse reward will feedback less information for learning. And if the reward is not straightforwardly evident (e.g. in the games with numerical feedback scores), we need also manually define one. However, in most cases an appropriate reward function is hard to design, this leads to the reward engineering problem [6]. In the past several decades researchers have developed many solutions like reward shaping [16], transfer learning [29], inverse reinforcement learning [1, 17], imitation learning [32], learning from demonstrations [22], etc.

Inverse reinforcement learning (IRL) refers to the problem of determining a reward function given observations of optimal behaviour [17], which is a learning from demonstrations method for solving the reward engineering problem. However most the IRL methods have lower data efficiency than the methods which learn directly from the demonstrations such as imitation learning or apprenticeship learning [1]. Imitation learning trains an agent to perform a task from expert demonstrations, with samples of trajectories from the expert, without having an access to the reward signal or interacting with the expert. Two main implementations for imitation learning are behavioral cloning (BC) and maximum entropy IRL (MaxEnt-IRL). BC is formulated as a supervised learning problem to map state-action pairs from expert trajectories to a policy [9] while MaxEnt-IRL solves the existing IRL problem that multiple policies can be inferred from a given set of demonstrations [32].

Recently [9] proposed an effective method, namely generative adversarial imitation learning (GAIL), to learn policies directly from data bypassing the intermediate IRL step. They showed a way to implement the supervised imitation learning using Generative Adversarial Networks (GAN) [7]. GAIL aims to recover only the expert policy instead of directly learning a reward function from the demonstrations. It relies on a dynamically trained discriminator to guide an apprentice policy to imitate the expert policy by optimizing the *Jensen-Shannon* divergence of the two policy distributions. The

Proc. of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), N. Agmon, M. E. Taylor, E. Elkind, M. Veloso (eds.), May 13–17, 2019, Montreal, Canada. © 2019 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

research showed that GAIL can learn more robust policies and fewer expert demonstrations are required for the training purpose.

Existing imitation learning methods show that the policy can be guided by the expert demonstrations instead of the reward signal. Generally they are not tabula rasa learning because the expert demonstrations are provided by humans. This leads the trained policies to be heavily biased toward to a human knowledge domain. The agent will perform similarly to the human's style even though some behaviours are sub-optimal or non-optimal. The potentially more powerful policies may be ignored in the training. In some scenarios it naturally assumes that the human expert demonstrations are available; however, most of the real-world applications have no available data sets, or no sufficient data for effective training.

In this paper we provide a novel general unsupervised learning framework of self-improving generative adversarial reinforcement learning (SI-GARL). The new framework avoids the reward engineering process based on the general policy iteration (GPI) [27]. It mainly contains two interleaved steps that conduct policy improvement and evaluation. We define a general policy improvement operator in the policy improvement step, then we can learn from a finite set of "generated expert" demonstrations produced by the operator instead of collecting the real human expert data as the guidance. In other words, we turn the policy improvement step to an imitation learning form, which uses the current policy as the prior knowledge to generate some improved policies and demonstrations. We implement this by embedding an imitation module to train the agent to mimic the produced demonstrations. The reward is not directly used in our framework. Instead, it is implicit in the policy improvement operator. In the policy evaluation step, the current policy network is rated by a metric of the difference between the current policy distribution and the improved policy distribution. Thus this again naturally can connect to GAN which trains a generative model to generate a distribution close to the given data distribution, by using a discriminator as a critic. The discriminator replaces the value functions/the rewards used in the traditional policy evaluation methods. Our imitation module adopt a GAN-like training method similar to GAIL.

DRL's result success depends heavily on how well we explore the solution space. And DRL highly rely on the rewards function or through observations. In the other hand, Imitation Learning uses supervised learning which is heavily studied with more stable behaviour. But the trained policy is only as good as the demonstrations from the experts, and IL will also experience an off-course drifting problem. Our framework combined imitation learning and DRL: The expert (improved policy) can tell us where to explore which save the DRL a lot of effort. And we apply DRL to refine a policy better than a human (because the generated policies are not from human) and able to handle the off-course situations better. In another view, the improvement operator can be seen as a policy space reducer which limited the exploration space to a subspace with higher quality policies. Comparing to the end-to-end DRL methods which learn directly from the observation-reward samples, our method adds an auxiliary guidance and makes the agent mimic to it. This significantly reduces the size of the exploration space and improves the sample efficiency. A end-to-end model free DRL (e.g. DQN) can be seen as a brain only uses intuitive thinking to fast react to the observations. The policy improvement in our framework can be seen as a brain thinks slower but plans longer and more explicitly based on the current intuitive thinking.

There are four main contributions in our work:

- We add an auxiliary loss to guide the model-free DRL. In our framework, agent will not only learn by the reward signal collected from the environment, but also imitate a generated improved policy, thus leading to better data efficiency.
- We enable the imitation learning method to work without the human demonstrations. The expert data is generated by the policy improvement operator, based on the policy output by the model-free DRL. The learning will not be limited or biased to any domain knowledge.
- The imitation part is implemented by adversarial training. After the policy improvement operator output a policy, we employ GAN to train the agent mimic towards to the improved policy. In another view, this is also an implementation of generalized policy distillation.
- We conduct a series of experiments to demonstrate the utilities of the proposed framework and compare the framework to the state-of-art RL methods. The experiments also include the investigations of the efficiency of the GAN method and the flexibility of the policy improvement operators.

2 PRELIMINARIES

A Markov Decision Process (MDP) is a mathematical system used for modelling decision making. We use a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ to define a finite Markov Decision Process (MDP). $\mathcal{S}$ denotes the state space, i.e a finite set of states. $\mathcal{A}$ denotes a set of actions the actor can take at each time step t. $P_a(s, s') = Pr(s_{t+1} = s' | s_t = s, a_t = a)$ denotes the probability that taking action a at time step t in state st will result in state s_{t+1}. $R_a(s, s')$ is the expected reward from taking action a and transitioning to s'. $y \in [1, 0]$ is the discount factor, which discounts the future reward.

Using an MDP, the goal is to determine which actions to take given a certain state. For this, a policy π needs to be determined which maps states s to actions a. A policy can either be stochastic $\pi(a|s)$ or deterministic $\pi(s)$.

For Sequential decision making problem, scenarios are broken up into a series of episodes. These episodes contain a set of rewards,states,and actions. Every episode contains n states corresponding to time t. These states are samples for the initial state s_0. For each time step t the actor chooses an action a_t based on a certain policy π. This policy is a probability distribution where the action is sampled given the state at time t, i.e. $\pi(a_t|s_t)$. Based on the action a_t, the environment will sample the reward R_t and the next state s_t according to some distribution $P(s_{(t+1)}, r_t | s_t, a_t)$. An episode runs for a predetermined number of time steps or until it reaches a terminal state.

To take the best actions according to the environment distribution, the policy π is chosen such that the expected reward is maximized. The expectation is taken over episodes τ containing a sequence of rewards, actions, and states ending at time $t = n$, i.e. the terminal state.

A policy maps states s to actions a. In this paper we focus on stochastic policies. A parameterized stochastic policy $\pi(a|s, w)$ is a action distribution with a model parameterized by w. The value function $V_\pi(s) = \mathbb{E}_\pi[\sum \gamma r(s,a)]$ of state s under policy π is the discounted expected total reward starting from state s. A policy π^* is optimal if $V_{\pi^*}(s) = \max_\pi V_\pi(s), \forall s$

3 BACKGROUND AND RELATED WORK

A general method, called generalized policy iteration (GPI), represents the core mechanism of most reinforcement learning techniques. It consists of two interactive processes. The *policy evaluation* step estimates the utility of the current policy π, that is, it computes V_π. The main purpose of this step is to gather information about the policy for computing the second step of the *policy improvement.* In this step, the values of the actions are evaluated for every state, in order to find any possible improvements. This step computes an improved policy π' from the current policy π using the information in V_π.

The closest related work is GAIL[9]. Our policy imitation part is similar to GAIL, which also uses GAN to imitate the expert. But their method needs a well collected and selected expert data from human. Their performance could be limited by the human level. Our framework does not rely on a high quality expert data, and more creative policies could be generated. Another related work is AlphaGO[24, 25], which used policy and value network as the estimators, and MCTS as a policy improvement operator. And at the initial training iterations AlphaGO learns from a database of human expert play, this may lead to policy biased to human knowledge. Our framework did not directly use the improved policies, instead it incorporated the GAN to mimic them. An initial investigation of the GAN imitation is shown in Table 2. SI-GARL shows better performance than SI-RL (which is a version more similar to AlphaGO) when they were facing to the same opponents.

Another related work is policy distillation[19, 20]. Distillation has been used in deep learning for compressing model and making model more stable. Policy distillation uses a well-trained agent to teach other random initialized agent. It can be cast as a supervised regression problem where the objective is to learn a model that matches the output distributions of all expert policies. Instead of calculating loss with maximum likelihood between teacher's action and student's action, it minimizes the Kullback-Leibler (KL) divergence with temperature τ. In our framework GAN is employed to distill the improved policy onto the initial policy at each iteration. Comparing to the existing policy distillation[19, 20], our method uses a self-generated policy as teacher. And we do not aim to compress the big network to a smaller one, but to feedback an incremental update to the current policy network.

4 SELF-IMPROVING GARL

In this section we start with a general framework of self-improving reinforcement learning (SI-RL) and present the basic settings of the networks and derive the update formula. Subsequently we introduce the new framework of self-improving generative adversarial reinforcement learning (SI-GARL) by adding GAN as the training module into the SI-RL framework. We elaborate three potential policy improvement operators in the SI-GARL framework.

4.1 General Framework

To generate the improved demonstration, we define a generic policy improvement operator I which maps a policy π and the state value V to an improved policy π'.

$$\pi' = I(\pi, V_\pi) \tag{1}$$

That is, the improved policy π' is claimed to be better than the direct output from the policy neural network. π and V can be considered as the prior knowledge to the operator I. The policy improvement operator does not have to be a specified structure. In our framework it is considered as a black box and can be any algorithm that can refine the initial intuitive strategy. Typically employing an operator will cost more computational resources by utilizing a current policy and state values multiple times. This is a trade off between a high quality prior knowledge which helps to shrink the exploration space and a consumption on computational resources. The goal is to seek a fixed point π' such that $\pi' = I(\pi', V_{\pi'})$ where the policy cannot be improved any further. A theorem guarantees the same convergence as GPI [27] when I satisfies the conditions that the policy order is defined.

Definition 4.1. [Policy Order] Let π and π' be any pair of policies such that for any s, if $V_\pi(s) > V_{\pi'}(s)$, the policy order is defined as $\pi \succ \pi'$

Definition 4.2. [Optimal Policy] For every MDP, there is at least one policy π^* that achieves the largest possible return from all states, and $\pi^* \succeq \pi$ for all the other policies π. This is called an optimal policy

THEOREM 4.3. *For any finite Markov decision process, given a policy π, if an operator I satisfies $I(\pi, V_\pi) \succ \pi$, then the sequence of policies generated by our policy iteration algorithm converges to a unique optimal policy π^**

PROOF. For a finite MDP (i.e. the state and action spaces are finite), policy iteration converges after a finite number of iterations. Since, for a finite MDP, the number of different policies is finite. Every policy $\pi_{k+1} = I(\pi_k, V_{\pi_k})$ is a strictly better policy than π_k unless in case $\pi_k = \pi^*$, in which case the algorithm terminates. □

By employing the policy improvement operator I we can produce the expert demonstrations $\tau_{\pi'}$ of the improved policies $\pi' = I(\pi, V_\pi)$ based on the prior knowledge π and V_π. Subsequently we transform the problem into an imitation learning task, which trains to match the agent policy to the improved policy. Similar to the objective function used in the policy distillation [20], we define the loss function as the KL-Divergence between the two policy distributions,

$$L_I = D_{KL}[I(\pi, V_\pi)||\pi] \tag{2}$$

where $D_{KL}[p||q]$ computes the KL-Divergence between the two distributions p and q.

In the new framework, the *policy evaluation* calculates a loss function defined as the KL-Divergence between the current policy and the improved policy. The loss is used as the utility function of the current policy: if the loss is large, the current policy still has potentiality to be improved. The *policy improvement* step trains to update the current policy in order to minimize the loss function.

We implement our framework by employing deep neural networks (DNN) as the function approximation. We use a DNN to estimate the state value function V as $\hat{V}(\theta, s) = DNN_\theta(s)$. $DNN_\theta(s)$ can be any type of DNN such as CNN, ResNet, etc. It is an end-to-end estimator with the input of a state $s \in \mathcal{S}$ and the output of the state value. θ represents the internal parameters of the DNN. Similarly, the policy estimator is defined as $\hat{\pi}(a|s, w) = DNN_w(s)$ where w represents the parameters in the DNN. By inputting the state s the DNN will generate an estimated policy distribution $\hat{\pi}(a|s, w)$ on the state s. We call the framework as the self-improving Reinforcement Learning (SI-RL) in Algorithm 1. The parameterized loss function and the update take the form as:

$$L_I(w, \theta, s) = D_{KL}[I(\hat{\pi}(\cdot|s, w), \hat{V}(\theta, s))||\hat{\pi}(\cdot|s, w)]. \tag{3}$$

Define $\pi' = I(\hat{\pi}(\cdot|s, w), \hat{V}(\theta, s))$ as the output of the policy improvement operator, the gradient can be derived:

$$\begin{aligned}
\nabla_w L_I(w_k, \theta_k, s) &= \nabla_w \mathbf{E}[\log \frac{\pi'}{\hat{\pi}(\cdot|s, w)}] \\
&= \nabla_w \sum_s \pi' \log \frac{\pi'}{\hat{\pi}(\cdot|s, w)} \\
&= \sum_s \nabla_w \pi' (\log \pi' - \log \hat{\pi}(\cdot|s, w)) \\
&= \sum_s \nabla_w \pi' \log \pi' - \sum_s \nabla_w \pi' \log \hat{\pi}(\cdot|s, w)) \\
&= -\pi' \nabla_w \log \hat{\pi}(\cdot|s, w) \\
&= -I(\hat{\pi}(\cdot|s, w), \hat{V}(\theta, s)) \nabla_w \log \hat{\pi}(\cdot|s, w)
\end{aligned} \tag{4}$$

Similarly, $\nabla_\theta L_I(w_k, \theta_k, s)$ can be derived as:

$$\nabla_\theta L_I(w_k, \theta_k, s) = -I(\hat{\pi}(\cdot|s, w), \hat{V}(\theta, s)) \nabla_\theta \log \hat{\pi}(\cdot|s, w) \tag{5}$$

Hence the update formulas for SI-RL will be:

$$\begin{aligned}
w_{k+1} &= w_k + \nabla_w L_I(w_k, \theta_k, s) \\
&= w_k - I(\hat{\pi}(\cdot|s, w), \hat{V}(\theta, s)) \nabla_w \log \hat{\pi}(\cdot|s, w)
\end{aligned} \tag{6}$$

$$\begin{aligned}
\theta_{k+1} = \theta_k - I(\hat{\pi}(\cdot|s, w), \hat{V}(\theta, s)) \nabla_\theta \log \hat{\pi}(\cdot|s, w) + \\
\nabla_\theta ||\hat{V}(\theta, s) - R)||^2
\end{aligned} \tag{7}$$

4.2 Training by the GAN method

Instead of optimizing the KL-divergence between the training policy and the improved policy produced by the policy improvement operator, we can integrate the GAN method [7] into the SI-RL framework to implement the training process of the imitation module. The main idea is to have two networks: discriminator and generator. The discriminator represents a classifier and assigns a low loss to the improved policies and a high loss to the initial ones. The goal of generator is to generate the initial policies so that their assigned loss is minimized. We keep using our policy network and allow it to update the parameters in the GAN training. Formally it tries to find the saddle point for the optimization below:

$$\min_w \max_D \mathbb{E}_{\pi_w}[\log D_\psi(s, a)] + \mathbb{E}_{\pi'}[\log (1 - D_\psi(s, a)] \tag{8}$$

where π_w is the policy model which we train, $\pi' = I(\pi, V)$ is the improved policy, and $D(a, s)_\psi$ is the discriminator network with the weight ψ. The generator will output an initial policy distribution, which is trained to predict what an agent will eventually decide to do given a state. This is similar to what GAIL [9] performs: to form an imitation module but to be guided by the dynamically generated improved policy. We call the GAN embedded SI-RL framework as the self-improving generative adversarial reinforcement learning (SI-GARL) in Algorithm 2. The policy network should be updated towards the natural gradient direction, while the discriminator and value network can be updated by the SGD methods.

In the practical implementation, we notice that the divergence used f-divergence can be chosen [18]. To deal with the mode collapse and vanishing gradient problems, we employ the $Wasserstein$ distance [2]. The policy network and value network can be set to share the latent layers' parameters to reduce the computational complexity such that they will not be updated separately. We show the engineering structure of the SI-GARL framework in Figure 1.

Algorithm 1 SI-RL

```
 1: while true do
 2:     {POLICY EVALUATION}
 3:     Produce improved policy
        $\pi'(\cdot|s) = I(\hat{\pi}(\cdot|s, w), \hat{V}(\theta, s))$
 4:     Calculate $L_I(w, \theta, s)$ by Eq. 3
 5:     if $L_I(w, \theta, s) < \epsilon$ then
 6:         STOP
 7:     end if
 8:     {POLICY IMPROVEMENT}
 9:     Update $w, \theta$ by Eq. 6,7
10:     Take action using updated $\hat{\pi}(\cdot|s, w')$
11: end while
```

Algorithm 2 SI-GARL

```
1: while true do
2:     {POLICY EVALUATION}
3:     Policy Evaluation step is the same as SI-GL
4:     {POLICY IMPROVEMENT}
5:     for i=0,1,2,... do
6:         Update $w$ by natural gradient: $\mathbb{E}_{\pi_w}[\nabla_w D_\psi(s, a)$ and $\psi$ by
           SGD on: $\mathbb{E}_{\pi_w}[\nabla_\psi D_\psi(s, a)] - \mathbb{E}_{\pi'}[\nabla_\psi D_\psi(s, a))]$
7:     end for
8:     Take action based on updated $\hat{\pi}(\cdot|s, w')$
9: end while
```

4.3 Analysis of GAN and Divergence

In this section we analyse the reason of using GAN instead of training directly by KL-devergence. Firstly we represent the loss function of GAN as below:

$$\begin{aligned}
L(w, D) &= \int_x \left(p_{\pi'}(x) \log(D(x)) + p_{\pi_w}(x) \log(1 - D(x)) \right) dx \\
&= \int_x f(D(x)) dx
\end{aligned} \tag{9}$$

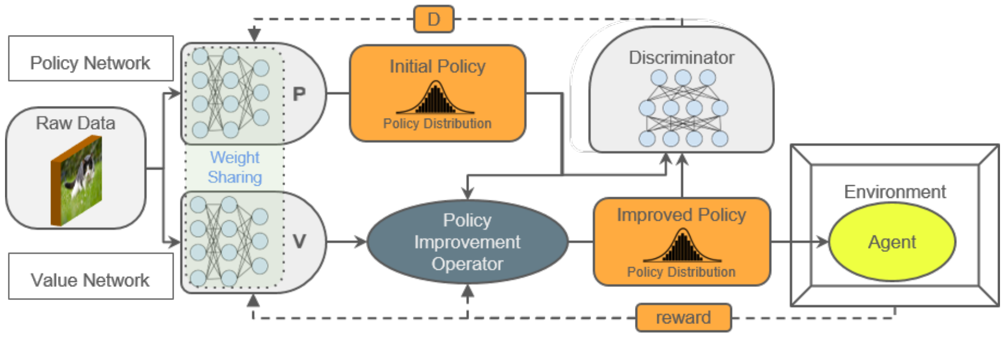

Figure 1: The architecture of SI-GARL framework. Arrows with solid lines show how the data transfer between modules, and those with dotted lines show the feedback signal used to update the networks. The orange blocks represent the internal data. The grey circle represents the policy improvement operator seen as a black box. The policy network and value network are the estimators of the policy ($P = \pi(\cdot|s)$) and value functions ($V = \hat{V}(s)$) as defined in Section 3.1 with, and the discriminator network is in the GAN training in Section 3.2.

We are interested in what is the best value of D to maximize $L(w, D)$, so we derive as the following:

$$\begin{aligned}\frac{df(D(x))}{dD(x)} &= p_{\pi'}(x)\frac{1}{ln10}\frac{1}{D(x)} - p_{\pi_w}(x)\frac{1}{ln10}\frac{1}{1-D(x)} \\ &= \frac{1}{ln10}\left(\frac{p_{\pi'}(x)}{D(x)} - \frac{p_{\pi_w}(x)}{1-D(x)}\right) \\ &= \frac{1}{ln10}\frac{p_{\pi'}(x) - (p_{\pi'}(x) + p_{\pi_w}(x))D(x)}{D(x)(1-D(x))}\end{aligned} \tag{10}$$

Then we set $\frac{df(D(x))}{dD(x)} = 0$, we get the best value of the discriminator:

$$D^*(x) = \frac{p_{\pi'}(x)}{p_{\pi'}(x) + p_{\pi_w}(x)} \in [0, 1] \tag{11}$$

When the generator is trained to its optimal, i.e. $p_{\pi'}(x) \approx p_{\pi_w}(x)$, the optimal discriminator should be $D^*(x) = \frac{1}{2}$. The loss function becomes:

$$\begin{aligned}L(w, D^*) &= \int_x \Big(p_{\pi'}(x)\log(D^*(x)) + p_{\pi_w}(x)\log(1 - D^*(x))\Big)dx \\ &= \log\frac{1}{2}\int_x p_{\pi'}(x)dx + \log\frac{1}{2}\int_x p_{\pi_w}(x)dx\end{aligned} \tag{12}$$

Recall the definition of Jensen-Shannon Divergence:

$$\begin{aligned}D_{JS}(p_{\pi'}\|p_{\pi_w}) =& \frac{1}{2}D_{KL}(p_{\pi'}||\frac{p_{\pi'} + p_{\pi_w}}{2}) + \frac{1}{2}D_{KL}(p_{\pi_w}||\frac{p_{\pi'} + p_{\pi_w}}{2}) \\ =& \frac{1}{2}\left(\log 2 + \int_x p_{\pi'}(x)\log\frac{p_{\pi'}(x)}{p_{\pi'}(x) + p_{\pi_w}(x)}dx\right) + \\ & \frac{1}{2}\left(\log 2 + \int_x p_{\pi_w}(x)\log\frac{p_{\pi_w}(x)}{p_{\pi'}(x) + p_{\pi_w}(x)}dx\right) \\ =& \frac{1}{2}\Big(\log 4 + L(w, D^*)\Big)\end{aligned} \tag{13}$$

Thus $L(w, D^*) = 2D_{JS}(p_{\pi'}\|p_{\pi_w}) - 2\log 2$. So optimising the generator model is treated as optimising the JS-divergence between the generative policy distribution and the improved policy distribution. Early research[10] speculates that one reason behind GAN's success is switching the loss function from asymmetric KL-divergence in MLE approach to symmetric JS-divergence. In our implementation, we further use the symmetric Wasserstein distance provides a smoother measure than JS-devergence to solve the problem of disjoint distributions [2].

4.4 Selections of Policy Improvement Operator

There are many choices for the policy improvement operator. It can traditionally be a policy based procedure or a value based procedure. It can also be a planning based or evolution based procedure. In this paper we adapt three typical options: trust region policy optimization (TRPO), Monte Carlo tree search (MCTS) and cross entropy method (CEM). We show how these operators can be efficiently embedded in our framework.

4.4.1 TRPO. Policy gradient (PG) methods [28] formulate a class of method to optimize a parameterized policy π_θ directly by maximizing expected return through a stochastic gradient ascent. TRPO is one of the notable policy gradient algorithms [23] and has a nice theoretical monotonic improvement guarantee.

Suppose we have a current policy π_0 that we wish to improve. We can write the performance of a new policy π in terms of the performance of π_0 $\eta(\pi) = \eta(\pi_0) + \mathbf{E}_{\rho_\pi}\mathbf{E}_{a\sim\pi(s)}[A_{\pi_0}(s, a)]$. [23] proved the following simpler bound involving KL-divergence between the new policy and the old policy: $\eta(\pi) \geq L_{\pi_0}(\pi) - Cmax_s D_{KL}(\pi_0(s)\|\pi(s))$ Where $C = \frac{2\gamma max_s |\mathbf{E}_{a\sim\pi'(s)[A_{\pi_0}(s,a)]}|}{(1-\gamma)^2}$. They used mean-KL divergence over state space as an approximation so that we can estimation it by $\bar{D}_{KL}(\pi_0\|\pi) = \mathbf{E}_{s\sim\rho_{\pi_0}}[D_{KL}(\pi_0(s)\|\pi(s))]$. Then the TRPO optimization problem become maximize$_\theta[L_{\theta_0}(\theta) - C\bar{D}_{KL}(\pi_0\|\pi)]$.

In the SI-GARL framework, we can directly use TRPO without modification as the policy improvement operator $I_{TRPO}(\pi_0, v) = TRPO(\pi_0, v,)$. Because of the monotonic improvement guarantee [23] in TRPO, it is obviously that TRPO satisfies Theorem 1. In practice, the estimated v values under the current policy π_0 can be used to calculate the approximation advantage $\hat{A}_{\pi_0}$, and then use conjugate gradient to approximate the gradient direction $F^{-1}g$, where, F is the Fisher Information Matrix equivalent to the second order derivative of the KL-divergence.

4.4.2 MCTS. Monte Carlo tree search [5] is a search algorithm that is based on a tree data structure, which can balance the exploration-exploitation dilemma, and performs effectively in the high dimensional search space. In theory MCTS can be applied to any domain that can be described in terms of $\{state, action\}$ pairs and simulations used to forecast outcomes.

The basic idea is to use a cycle of Selection, Expansion, Simulation, and Back-Propagation to build and selectively explore a tree, where each node corresponds to a state and each edge from a node corresponds to an action. Leaf nodes are chosen according to a particular algorithm, and then full games are simulated from those nodes until they terminate. Then, the results of the simulation are back propagated up the tree all the way to the root node. And, unlike other RL techniques, MCTS does not require that the full state space be enumerated, which may be necessary to fully estimate the value of a state in terms of its future rewards.

MCTS is often used as an online planning method. *AlphaGO* and *AlphaGO Zero* have shown success on GO game by combining RL with MCTS [24, 25]. We use MCTS in the SI-GARL framework as a planning-based police improvement operator. It provides a long term panning ability compared to the policy gradient methods. The Upper Confidence Bounds for Trees (UCT) algorithm is a particular instance of MCTS with the formula $UCT = \bar{X}_j + 2C_p\sqrt{\frac{2\log n}{n_j}}$, where $\bar{X}_j$ is the average reward of child j, n is the number of times the parent node has been visited, n_j is the number of times child j has been visited, and $C_p > 0$ is a constant. We modify this formula to fit the SI-GARL framework: $UCT_\pi = v_j + 2C_p\sqrt{\frac{2\log n}{n_j}} + k\frac{\hat{\pi}(a|s_j)}{n_{s_j,a}+1}$. We use the estimation value v_j of the states replace the average reward $\bar{X}_j$, and add a policy term to make UCT_π be able to improve the policy by utilising the policy produced by the current network. It has been proved that MCTS converges to an optimal solution [11], such that the policy improvement operator $I_{MCTS}(v, \pi_0) = MCTS(v, \pi_0)$ under UCT_π satisfies Theorem 1.

4.4.3 CEM. In the SI-GARL framework the policies are always generated by the network. Hence they can all be seen as parametrized policies with the network parameters. To solve the policy optimization problem of maximizing the total (discounted) reward given some parametrized policy, a simple approach is the derivative free optimization (DFO) which considers this problem as a black box with respect to the parameter θ, which is different from the policy gradient method. One particular DFO approach is called the cross-entropy method (CEM). At any point in time, it maintains a distribution over parameter vectors and moves the distribution towards parameters with a higher reward.

To adapt the CEM in the SI-GARL framework, we keep changing the parameters in the network: given a current policy network π_0, we initialize μ as the current network parameters. Then run CEM to update the parameter in the policy network $I_{CEM}(v, \pi_0) = CEM(\pi_0)$, which is similar to the evolution strategy algorithm [21]. The implementation includes the following four steps:

5 EXPERIMENT

In this section we investigate our framework in a series of experiments to answer the following questions:

- Does the self-improving imitation learning framework make the performance better than that by training only by the policy improvement operator which the framework employed?
- For the scenarios without clearly defined reward signal, can the SI-GARL learn better policies than the standard DRL baselines like DQN and A3C? And how important does GAN play a role in the training process? This will help us answer why GAN is needed when a parameterized policy is already obtained from the policy improvement operator.
- For the scenarios with clearly indicated reward signal, can the SI-GARL still reach or even outperform the state-of-art RL algorithms? And how does the selection of the policy improvement operator impact the performance for various domain problems?

The experiments on the first Gomoku section compare against the black-box policy improvement algorithms. While that on miniRTS and Atari sections are compared to standard DRL baselines. In miniRTS section there is also a comparison between our GAN version SI-GARL and non-GAN version SI-RL.

We develop a light version and a whole version of the SI-GARL framework to adapt different scenarios. Normally the light version generative network uses a 30 layer's fully-connected network with 3 extra layers of end nodes to output the policy or value. The whole version has 80 layers. There are 3 fully convolution layers using 32, 64 and 128 3×3 filters at the beginning. The *ReLu* activation function is employed. In the policy output end, four 1×1 filters are used to do the dimensionality reduction. After that there is an extra full connection layer. The $softmax$ nonlinear function directly outputs the probability of each action on the state. At the value end, we first connect two 1×1 filters, then a fully connected layer of 64 neurons. It uses the tanh nonlinear function to output the evaluated score in the range $[-1, 1]$.

5.1 Gomoku

We chose Gomoku as the first experiment platform since it is a zero-sum game and the state space is not as large as Go. We choose MCTS as the policy improvement operator in this experiment. To reduce the computational time, we test a small size 8×8 chessboard with five chess. A player who first makes a line of 5 chess win the game. We use 4 binary 8×8 feature matrix. The first two are the board states of the current and the opponent players. The third matrix indicates the position of the last move of the opponent. The fourth matrix indicates which player can move at this step. The self-play data was generated directly from the current network and was used to train and update itself. Using the latest model to generate self-play data will also benefit the exploration. We add Dirichlet noise $P(s, a) = (1 - \varepsilon)P_a + \varepsilon\eta_a$, similar to the way of exploring in deterministic policy gradient methods. The parameters of Dirichlet noise are 0.3, $\boldsymbol{\eta} \sim \mathrm{Dir}(0.3)$ with $\epsilon = 0.25$.

A total of 3050 games were played in this experiment. The loss changed from around 4.0 down to around 2.2. We observe the change of entropy of the policy during the training. The policy network will learn which positions should have a large probability of moving in different situations. The distribution will become strong bias, thus the entropy will be smaller. Figure 2 shows that a loss function with the number of self-play matches changes on 8×8 broad, and the changes of the entropy of the policy network output observed in the same training process. The results show that the SI-GARL framework is on the right learning path.

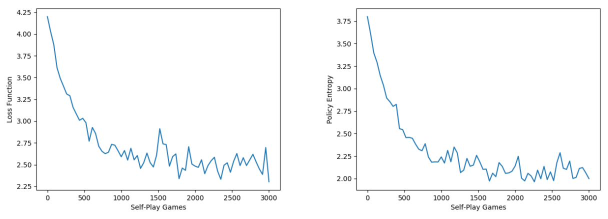

Figure 2: Loss and entropy change in training

In addition, we set a competition to observe the performance of the training network. From the beginning, every 50 self-play games we use our latest network to fight versus a single MCTS planning method which is the same as the policy improvement operator we employed in our framework. We set a different level of the MCTS AI by changing the number of MC simulations on each step, i.e. $MCTS_N$ means for each step the AI will do N simulations to estimate the future. When a level of the AI was defeated to 10: 0 by our trained AI model, it will be upgraded into the next level by increasing N and so on. While our trained AI model employs a $MCTS_{400}$ operator. After the above 3050 self-match training experiments, we observed that it indeed increases the performance while it keeps beating higher and higher level' opponents in Table 1.

Number of Games	Score(w:d:l)	Opponent level
100	3:1:6	$MCTS_{1000}$
450	10:0:0	$MCTS_{1000}$
700	7:1:2	$MCTS_{2000}$
1100	10:0:0	$MCTS_{2000}$
1250	8:0:2	$MCTS_{3000}$
1600	10:0:0	$MCTS_{3000}$
2300	8:2:0	$MCTS_{4000}$
3050	9:1:0	$MCTS_{4000}$

Table 1: Comparison between SI-GARL and MCTS opponents. The score shows the counts of win, draw and loss in 10 games.

5.2 miniRTS

In this experiment we train our model in a more complex environment without obvious reward functions. Game environments are widely used to test novel reinforcement learning algorithms. The real-time strategy (RTS) games realistically simulate real-world scenarios. ELF is an platform for game and reinforcement learning research, which gives an end-to-end solution from game simulators to training paradigms [31]. It requires less resources compared to other RTS simulators. The complexity of the RTS game always challenges design of the reward function and this is exactly what the SI-GARL framework tries to solve. As the game engine focuses on two-player games, we choose CEM and MCTS as the policy improvement operators for both SI-RL and SI-GARL in the experiments. There are two stages in the experiment. In the first stage,

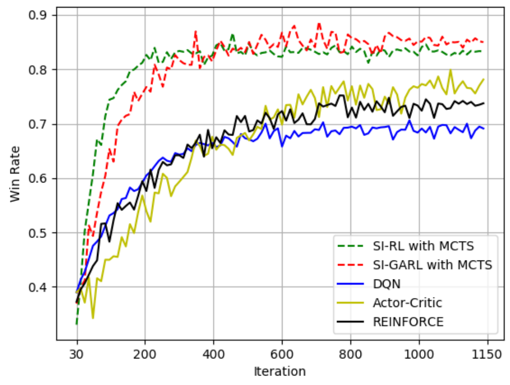

Figure 3: Compare our two methods to DQN, REINFPRCE and A3C on the win rate to a rule-based AI in training.

because the MiniRTS in ELF provides a rule-based simple AI player as a baseline opponent, we train 5 methods including SI-GARL (MCTS), SI-RL (MCTS), DQN, REINFORCE and A3C to fight against the rule-based AI. Since RTS is much more complex than Gomoku, we use the whole version network to improve the learning ability. The win rate curves in the training in Figure 3 show how different algorithms perform versus a baseline AI.

In the second stage, after all the methods have been trained by a default rule-based AI as opponent, we change the opponent to each of the AIs well-trained from the first stage. Then we evaluate our AIs using SI-GARL and SI-RL via 10000 game simulations on each of the other opponents. The accumulated win rates are shown in Table 2. We notice that when the opponent is a rule-based AI, the importance of using GAN is not significant. However when our models face more complex and intelligent opponents, GAN adds more values to the performance improvement. Especially SI-GARL reaches over 60% win rate to directly beat SI-RL.

Opponent	**SI-RL**	**SI-GARL**
Rule-based	82.7%	84.1%
DQN	68.3%	73.4%
REINFORCE	64.5%	71.8%
A3C	63.8%	69.9%
SI-RL	NA	60.2%

Table 2: Comparison between SI-GARL and SI-RL without the GAN training. The accumulated win rates are listed.

GAME ROM	SI-GARL (MCTS)	SI-GARL (TRPO)	SI-GARL (CEM)	UCT	A3C	DQN
Amidar	**(423.5)**	266.7	121.4	187.2	278.6	125.4
Assault	1635.1	3319.9	1637.1	1498.7	**(3715.7)**	3103.6
Battle Zone	12131.3	8904.5	**(19021.5)**	9162.4	10712.4	16605.6
Beamrider	3014.6	10314.7	806.5	2896.1	**(13176.8)**	8557.1
Breakout	**(636.7)**	501.4	17.4	404.7	560.4	247.3
Centipede	5016.9	2215.3	**(7147.9)**	4102.3	3477.9	3678.1
DemonAttack	69385.3	**(81353.4)**	2639.5	57893.3	79674.2	11376.5
Freeway	22.1	23.9	**(32.2)**	24.1	26.8	25.3
Gravitar	367.1	334.8	**(917.4)**	397.1	274.6	226.3
Kangaroo	1677.3	1104.2	**(9759.3)**	1879.4	1146.7	2489.3
Phoenix	12634.6	20456.3	5867.1	6871.5	**(27914.6)**	9768.4
Pong	**(20.9)**	17.4	20.8	20.6	19.3	16.8
Q-bert	**(33756.7)**	21793.0	877.4	15321.2	14002.5	4557.2
RiverRaid	5837.4	**(11419.8)**	7032.6	4749.9	9469.1	3885.9
RoadRunner	**(33891.4)**	31046.3	18934.6	29742.8	32168.3	9127.4
Seaquest	2067.3	2104.5	1407.3	2018.4	2267.1	**(2749.6)**
Skiing	12493.6	12135.8	**(17072.4)**	10024.2	13367.1	8463.5
StarGunner	**(76345.1)**	62123.4	1931.4	33756.1	57193.6	31482.7
Venture	487.3	98.4	**(658.3)**	212.4	22.4	42.7
Zaxxon	**(6794.5)**	3036.7	6191.4	3631.7	2979.2	811.4

Table 3: Final results obtained by SI-GARL with different policy improvement operators on 20 Atari 2600 games, and compare to two RL methods A3C, DQN and a planning based policy search method UCT

5.3 Atari

A popular benchmark for evaluating reinforcement learning has been on the Atari game domain, provided through the Arcade Learning Environment (ALE) - an object-oriented framework that allows researchers to easily develop AI agents [4]. To save the experimental times, we used ELF which has incorporated ALE with the support of paralleled running on multi-cores.

The standard evaluation metric for Atari game is the score returned by the game itself. This means Atari game can be easily evaluated by an obvious numerical reward score. We test our framework on 20 selected Atari games to see if the performance can reach the same or above that of the three benchmark methods UCT, A3C and DQN. Different from the SI-GARL framework, all these benchmarks learn directly based on the reward score. We also investigate how the three implemented policy improvement operators perform across the 20 different games. We used the same A3C and DQN setup and followed the images pre-processing in [13]. The setup of UCT is the same as that in [8]. Our policy network used a whole version of the SI-GARL framework with well tuned hyperparameters. All the games were trained for 160 million frames on our three methods and 640 million frames on the benchmarks.

The averaged scores over the last 20 episodes are shown in Table 3. Most of the best scores are reached by the SI-GARL with fewer training frames. In most times our method can lead to the close or better game scores while it does not directly optimize the scores. We notice that there is no winner between our three methods for all games. For example, some games relying on planning would require a MCTS based policy improvement operator. Our framework has the flexibility for different types of tasks.

We also have computational time cost for each method. The computational cost varies in different settings because the improvement operator performance varies widely on multiple games. SI-GARL with MCTS costs roughly 5-8 times compared to DQN, while with TRPO costs 3-4 times and with CEM costs 4-6 times. Overall the new framework endorses good performance in terms of computational cost.

6 CONCLUSION

This paper proposes and implements a novel SI-GARL framework that avoids to face directly to the reward engineering problem in RL. We define a policy improvement operator to provide flexibility to the SI-GARL framework, and employ the operator as a black box to implement the self-improving procedure. We integrate GAN into the SI-GARL framework in order to further improve the exploration quality and the data efficiency. The imitation step is implemented by adversarial training, which is not a standard GAN, but similar to GAIL. The generator-and-discriminator gaming principle still exists in our framework. The framework does not seem to be similar to a classical GAN, but inherits its essential idea. We show in the experiments that it is worth to spend more computational cost on this framework due to its outstanding performance. Both GAN and self-improving procedures show their potential in multiple test domains. In the future we will turn to investigate on how to select the proper improvement policy operator and develop a meta-selector.

ACKNOWLEDGMENTS

The work is supported by the UKRI - EPSRC New Investigator Award under EPSRC Reference: EP/S011609/1.

REFERENCES

[1] Pieter Abbeel and Andrew Y Ng. 2004. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*. ACM, 1.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*. 214–223.

[3] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. 2017. A Brief Survey of Deep Reinforcement Learning. *CoRR* abs/1708.05866 (2017). arXiv:1708.05866 http://arxiv.org/abs/1708.05866

[4] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47 (2013), 253–279.

[5] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games* 4, 1 (2012), 1–43.

[6] Daniel Dewey. 2014. Reinforcement learning and the reward engineering principle. In *2014 AAAI Spring Symposium Series*.

[7] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[8] Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard L Lewis, and Xiaoshi Wang. 2014. Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning. In *Advances in neural information processing systems*. 3338–3346.

[9] Jonathan Ho and Stefano Ermon. 2016. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*. 4565–4573.

[10] Ferenc Huszár. 2015. How (not) to train your generative model: Scheduled sampling, likelihood, adversary? *arXiv preprint arXiv:1511.05101* (2015).

[11] Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*. Springer, 282–293.

[12] Nate Kohl and Peter Stone. 2004. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, Vol. 3. IEEE, 2619–2624.

[13] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*. 1928–1937.

[14] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[15] Andrew Y Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. 2006. Autonomous inverted helicopter flight via reinforcement learning. In *Experimental Robotics IX*. Springer, 363–372.

[16] Andrew Y Ng, Daishi Harada, and Stuart Russell. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, Vol. 99. 278–287.

[17] Andrew Y Ng, Stuart J Russell, et al. 2000. Algorithms for inverse reinforcement learning.. In *Icml*. 663–670.

[18] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. 2016. f-gan: Training generative neural samplers using variational divergence minimization. In *Advances in Neural Information Processing Systems*. 271–279.

[19] Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. 2015. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342* (2015).

[20] Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. 2015. Policy distillation. *arXiv preprint arXiv:1511.06295* (2015).

[21] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. 2017. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864* (2017).

[22] Stefan Schaal. 1997. Learning from demonstration. In *Advances in neural information processing systems*. 1040–1046.

[23] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International Conference on Machine Learning*. 1889–1897.

[24] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484–489.

[25] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *Nature* 550, 7676 (2017), 354.

[26] Satinder Singh, Diane Litman, Michael Kearns, and Marilyn Walker. 2002. Optimizing dialogue management with reinforcement learning: Experiments with the NJFun system. *Journal of Artificial Intelligence Research* 16 (2002), 105–133.

[27] Richard S Sutton and Andrew G Barto. 1998. *Reinforcement learning: An introduction*. Vol. 1. MIT press Cambridge.

[28] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*. 1057–1063.

[29] Matthew E Taylor and Peter Stone. 2009. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research* 10, Jul (2009), 1633–1685.

[30] Gerald Tesauro. 1995. Temporal difference learning and TD-Gammon. *Commun. ACM* 38, 3 (1995), 58–68.

[31] Yuandong Tian, Qucheng Gong, Wenling Shang, Yuxin Wu, and C Lawrence Zitnick. 2017. Elf: An extensive, lightweight and flexible research platform for real-time strategy games. In *Advances in Neural Information Processing Systems*. 2656–2666.

[32] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. 2008. Maximum Entropy Inverse Reinforcement Learning.. In *AAAI*, Vol. 8. Chicago, IL, USA, 1433–1438.